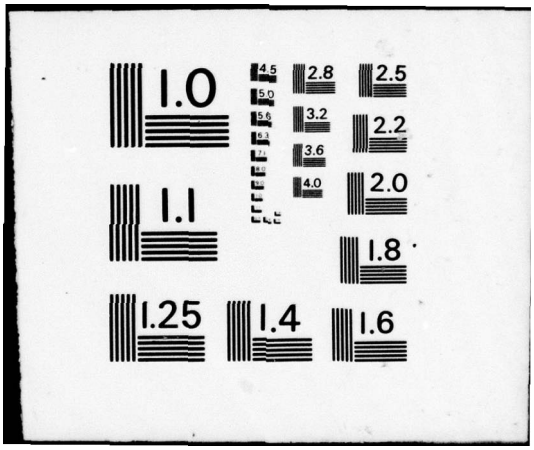
AD-A035 397 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
THE REFORMULATION MODEL OF EXPERTISE.(U)
DEC 76 W S MARK
UNCLASSIFIED MIT/LCS/TR-172 N00014-75-C-0661
NL

1 OF 3
ADA035397


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-172

THE REFORMULATION MODEL OF EXPERTISE

William S. Mark

This research was supported by the
Advanced Research Projects Agency
of the Department of Defense and
was monitored by the Office of Naval
Research under Contract No. N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TR-172
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): The Reformulation Model of Expertise
5. TYPE OF REPORT & PERIOD COVERED: PhD thesis, Sept., 1976
6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TR-172
7. AUTHOR(s): William Scott Mark
8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0661
9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technology Sq., Cambridge, MA 02139
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS:
Advanced Research Projects Agency
Department of Defense
1400 Wilson Boulevard
Arlington, Virginia 22209
12. REPORT DATE: Dec[illegible] 1976
13. NUMBER OF PAGES: 256
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
Office of Naval Research
Department of the Navy
Information Systems Program
Arlington, Virginia 22217
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report):
Approved for public release; distribution unlimited
Doctoral thesis
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
expert systems, consulting programs, automatic programming, artificial intelligence, management information systems
20. ABSTRACT (Continue on reverse side if necessary and identify by block number):
This research develops a methodology for implementing a class of expert problem-solving programs which must create models of the problems they are given before they can apply their expert knowledge to those problems. That is, given a problem in their domain of expertise, these programs "reformulate" the problem description in terms of their built-in abstract models of the domain. The implementation methodology described in this report provides the following capabilities:

(cont'd on p 1473B).

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

409 648

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(20, cont.)

(1) pieces of expert knowledge can be added to the program in a fairly independent manner without regard to their control structure implications;

(2) programs can delve through irrelevant information in the problem description to pick out only the facts needed for problem-solving;

(3) programs can model the input problem at various levels of detail, as required by the problem-solving task; and

(4) programs can tailor their expert models to the problem at hand.

This is accomplished by controlling the general procedure which maps pieces of the expert model into pieces of the input with feedback information taken from the model of the input so far.

The methodology is demonstrated with a working program which acts as a business consultant dealing with firms which have workforce control problems. Input is in the form of several paragraphs of simplified English which the program models in accordance with a cause-effect flow model of the firm. If the modelling effort is successful, policy suggestions are made which improve the performance of the firm in question.

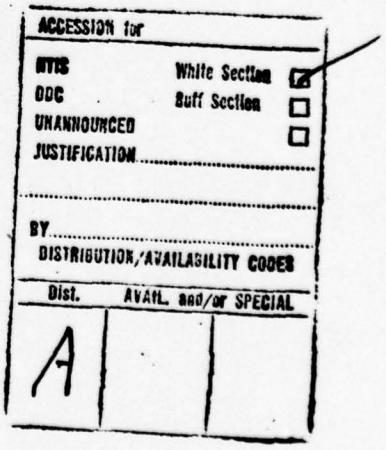
MIT/LCS/TR-172

THE REFORMULATION MODEL OF EXPERTISE

William Scott Mark

December 1976

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract Number N00014-75-C-0661.



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE
(formerly PROJECT MAC)

CAMBRIDGE MASSACHUSETTS 02139

THE REFORMULATION MODEL OF EXPERTISE

by

William Scott Mark

ABSTRACT

This research develops a methodology for implementing a class of expert problem-solving programs which must create models of the problems they are given before they can apply their expert knowledge to those problems. That is, given a problem in their domain of expertise, these programs "reformulate" the problem description in terms of their built-in abstract models of the domain. The implementation methodology described in this report provides the following capabilities:

- **<> pieces of expert knowledge can be added to the program in a fairly independent manner without regard to their control structure implications**
- **<> programs can delve through irrelevant information in the problem description to pick out only the facts needed for problem-solving**
- **<> programs can model the input problem at various levels of detail, as required by the problem-solving task**
- **<> programs can tailor their expert models to the problem at hand**

This is accomplished by controlling the general procedure which maps pieces of the expert knowledge base into pieces of the input with feedback information taken from the model of the input so far.

The methodology is demonstrated with a working program which acts as an expert business consultant dealing with firms which have workforce control problems. Input is in the form of several paragraphs of simplified English which the program models in accordance with a cause-effect flow model of the firm. If the modelling effort is successful, policy suggestions are made which improve the performance of the firm in question.

This report is a minor revision of a thesis of the same title submitted to the Department of Electrical Engineering and Computer Science of the Massachusetts Institute of Technology on August 9, 1976 in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

Acknowledgements

I would like to thank Bill Martin for his supervision and assistance throughout this research; Gerry Sussman, Rand Krumland, Gretchen Brown, and Bill Long for careful reading and suggestions at various stages; Bob Baron for those TECO macros; and Marvin Minsky, without whose influence much of this thesis would not have been written.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract Number N00014-75-C-0661.

This report is a minor revision of a PhD thesis submitted to the Department of Electrical Engineering and Computer Science of the Massachusetts Institute of Technology.

Table of Contents

List of Figures

Chapter I

In which the Hero is Introduced and Differentiated from his Fellows

In recent years, a considerable amount of computer science research has been directed toward the development of programs which provide problem-solving expertise to the user. The goal of this research is the production of computer programs which fill the role of a consultant in their areas of expertise. The creation of such programs is an enormous research effort, encompassing problems of natural language understanding, knowledge representation, and other research problems in artificial intelligence and computer science in general.

This thesis is about constructing expert systems for a limited class of expert problem-solving processes. In particular, I define the class of *reformulation* processes to be those in which the expert must develop a model of the input problem description before he can apply his expertise (analytic and debugging tools, etc.) to it. The research is predicated on the following model of this class of expert problem-solving activities:

> The expert has a preconceived, well-developed model of "the way things work" in his domain of interest. His method of solving problems is to *cast the problem situation in terms of his own model* (which I call the "abstract model"). The expert can then use his well-developed knowledge of how the *model* works to solve the problem at hand.

The important thing to realize is that the expert models *in order to understand*. He is coming to grips with a complex situation by casting it in terms of a theory of "how things work" which he already understands (he knows how to solve problems stated in terms of it, he knows what can go wrong in it, etc.).

I call this process of modelling the problem to enable the application of expert

knowledge *reformulation*. The expert works by taking the existing formulation of the situation and *re*formulating it into his own concept structure. In other words, in order to solve a problem, the expert must construct an abstract model version of the input problem description:

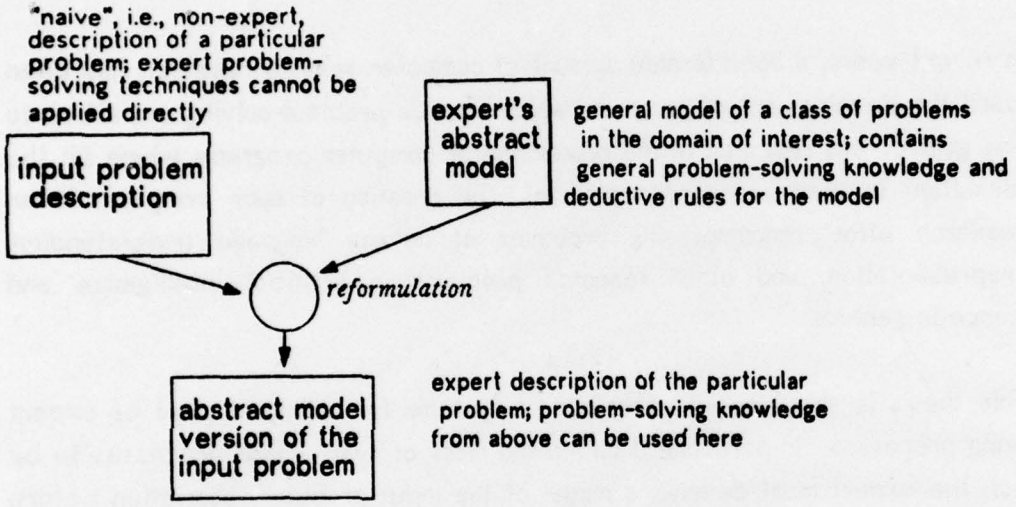


Figure 1. Reformulation

The process of constructing the mapping between the appropriate parts of the abstract model and the input problem description to create the "abstract model version" of the input problem is thus the reformulation.

The implementation methodology I develop in my thesis takes advantage of the above reformulation problem-solving structure in several ways. First, *the needed mapping for any given problem is constructed by transforming the relevant pieces of the abstract model* so that they can be matched with the appropriate pieces of the input. That is, the onus of map construction is placed on procedures which know how to handle the well thought-out and canonical abstract model structures rather than on procedures which must deal with the more freely expressed input structures. Second, the implementation methodology

stresses that these *map construction procedures should be solely responsible for selecting and utilizing the abstract model knowledge needed to solve any particular problem.* This enables the expert system builder to attach all of his expert knowledge about a concept to that concept without having to worry about the control structure issues of how that information is accessed and used by the system to solve any particular problem. Finally, *the map construction procedures' decision as to which abstract model pieces are relevant to a problem and how they should be transformed to create the needed mapping to the input is controlled via features of the abstract model version which is being built for that specific problem.* In this way the process is driven neither strictly by the input, which contains much information which is irrelevant to the expert model and may send the system off on a tangent, nor strictly by the abstract model, which is much more general than any given problem and may thus involve the system in considerations which are unnecessary for that problem. The abstract model version of a problem always contains the system's most up to date information about how its model has been applied to that problem so far. This information, in the form of features extracted from the modelled version, is placed in a special data structure which is used to control the effort of the map construction procedures on the problem being modelled. This special structure makes clear the control implications of the selected pieces of the modelled version. Three such implication categories are distinguished: features which affect the entire modelling effort, features which affect only local piece-to-piece connectivity, and features which affect the way in which closely interacting groups of pieces should be modelled, but do not affect other such groups. The feature-extraction process which produces the special data structure is also domain-independent--i.e., it does not depend on the interpretation of the concepts in the abstract model. The use of this special structure with its three groups of modelled version features constitutes the system's basic problem-solving control structure.

This implementation methodology presents a new way to organize expert knowledge in computer programs to make effective expert systems. It is therefore to be compared with other approaches to building knowledge into programs such as the "frame" notion ([Kuipers], [Minsky], [Rubin]), the modified production rule scheme of MYCIN [Shortliffe], ARSE [Sussman and Stallman], etc. and the general network approach ([Fahlman (May, 1975)], [Rieger]). In general, the approach I take in my thesis is designed to allow the expert model to be constructed in a such a way that the knowledge of each part is fairly independent of other parts (to make it easier to build the model) while at the same time providing enough control structure to make the system a useful, efficient problem-

solver even in a large domain. Thus, it is designed to avoid forcing the expert to have to worry about how each piece of the model fits in with related pieces to provide the overall control structure of the system (as is necessary in frame systems). Furthermore, it is designed to avoid the exhaustive search, questionnaire control structure which is the cost of piece independence in the MYCIN system (which seems to be fundamentally domain-limited). Finally, it is designed to be more structured and efficient, and certainly more easily implementable with current hardware technology, than the distributed knowledge approaches suggested by Fahlman and Rieger.

We can see some of the basic features of the way the reformulation process is implemented by the methodology described above by considering the following mini-example: Suppose that an expert system is designed to act as a business consultant whose abstract model consists of a view of the firm as a flow process which can be analyzed by the techniques of control theory. In particular, the system's model of a hiring policy in a firm is that it is a control function in a feedback loop which is continuously trying to eliminate any discrepancy between the number of people currently employed by the firm and the number of people the firm *wants* to employ at that time:

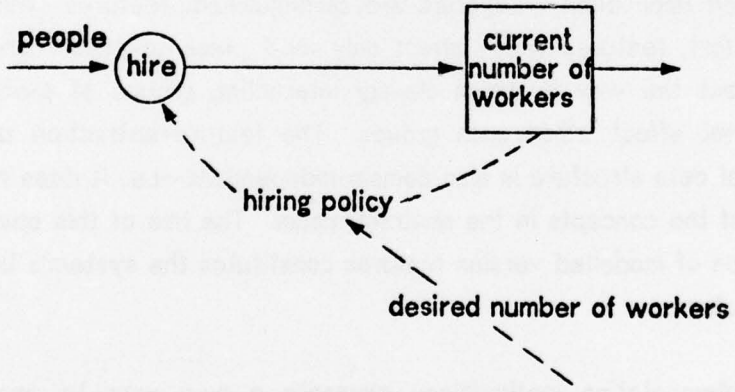


Figure 2. Expert model of hiring

In the representation scheme used in this thesis, this hiring policy would look like

```
(*)  (SMOOTH (DISCREPANCY
       ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED))
       (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))).
```

Given a problem description of a firm, say one that was having labor problems of some sort, the task of the business consultant expert system would be to reformulate the firm's hiring policy in terms of the above abstract model of hiring.

The problem is that the description of hiring in the input is not going to look anything like figure 2. The manager doesn't think of hiring in terms of a feedback control mechanism. Instead, he may describe his hiring policy in terms of something that controls the production rate of the firm, or, as in the problem description we will explore later in this thesis, with the statement

> Hiring has taken place when the order backlog became large and sales were being lost to competitors.

or, in the program's parse form:

```
(**)  ((DEPENDS-ON
         (ACT-ON (NUMBER-OF (PEOPLE) (INCREASED))
                 (HIRE))
         (STATE-OF ((ORDER (BACKLOGGED)) OF (CUSTOMER))
                   (LARGE)))
               REASON ((ACT-ON (ORDER) (TRANSFER))
                       DESTINATION (COMPETITOR)
                       AGENT (CUSTOMER)))
```

That is, in the manager's hiring policy for this problem description, "hire more people when the backlog becomes large", hiring is seen as a mechanism for controlling the size of the backlog of customer orders in order to avoid losing sales.

Now the consultant's model embodied in the expert system cannot deal with the above hiring policy stated in the manager's terms: it cannot analyze the manager's statement for defects, or reason about how it might affect other parts of the firm. Before it can solve any labor problems in the firm, it must create an abstract model version of the firm's hiring policy. That is, the program must model (**) in terms of (*); in other words, construct a mapping between (*) and (**).

How can it do this? As I said earlier, the first tenet of the implementation methodology of this thesis is that it is (*) which should be transformed into a structure which models (**), not vice versa. The reason for this is that the expert model, including (*), is presumably more abstract than the manager's description: (*) is designed to be applied to a wide variety of hiring policies described at the level of (**). By concentrating on changing (*), the map construction process can work from a smaller set of transformations. Furthermore, since the abstract model is well worked out and is built into the expert system, more is known about its concepts than those of the manager description--thus it is easier decide what transformations can be made on the concept structures without losing information or creating objects which the deductive procedures of the model cannot work on.

Therefore, the program sees the map construction problem as one of applying a series of transformations to (*) until it matches (**). The implementation methodology has two basic ways of attacking this problem: applying structural transformations to change the way in which (*) is expressed, and utilizing alternative or related constructs dealing with hiring which the expert provides as part of his abstract model to change (*) into something which is closer to the input description. The distinction between the two different kinds of transformations is reflected in the two different kinds of elements which can appear in the abstract model representation. There are structural concepts (like SMOOTH, DISCREPANCY, NUMBER-OF, NIL and MODIFICATION above) that relate one abstract model concept to another and can be used to construct many different abstract models. Then there are the actual concepts that the expert uses to express knowledge in his domain (like PEOPLE, EMPLOYED, and DESIRED above). The program's structural transformations apply to the first kind of element. For example, SMOOTH, which represents a time-averaging function, can be transformed into a variety of formulations to model the way in which the manager looks at a particular variable over time. If it turned out in this case that the manager observed changes in the workforce level averaged over a one week

period, SMOOTH would be transformed into a first order averaging delay with a one week time constant.

More pertinent to the problem of matching (*) to (**) is the fact that the program can make use of alternate formulations of (PEOPLE (EMPLOYED)) (i.e., "employees") and ((PEOPLE (EMPLOYED)) MODIFICATION (DESIRED)) and related concepts to construct the desired mapping. Unlike the structural transformations, these alternate formulations are provided by the expert as part of the abstract model. However, it is important to understand that they are not provided with any intent or restriction on how they are to be used in mapping. The expert simply records his various models of a particular concept, i.e., the various forms that concept can take in the abstract model, (along with any other knowledge about that concept--necessary characteristics, etc.) as properties of that concept. These can then be used by the mapping process either directly, for substituting one alternate for another in order to more closely match the manager's way of expressing a concept, or indirectly, to make inferences about how that concept affects other parts of the model. For example, the expert may provide the fact that sometimes extra people are needed to keep the backlog at a desired level:

```
((PEOPLE (EMPLOYED)) MODIFICATION (EXTRA))
    (RATIO
      (SMOOTH (DIFFERENCE
                (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                ((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                    MODIFICATION (DESIRED))))
      (PRODUCTIVITY))
```

(i.e., "one model of extra personnel is that they are the number of employees needed to maintain the order backlog at a desired level--the number given by the ratio of the backlog gap, the amount by which the current backlog exceeds the desired backlog level, and the individual productivity of the workers"). However, the expert does not have to say how to use this model to substitute for ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL) MODIFICATION (DESIRED)) in (*), as is necessary in this case, or how to use it inferentially to show the connection between backlog and workforce fluctuation, as is necessary in other cases--this is taken care of by the transformation procedures which are built into the expert system. Therefore, the expert is not asked to worry about the control

structure implications of the information; he just places what he knows about a concept on properties of that concept.

Finally, as I said before, the transformation procedures which utilize the structural transformations and expert-provided features of the model are general, i.e., not abstract model specific. Therefore, their efforts must be controlled by features of the particular abstract model and the particular problem description under consideration for each problem-solving effort. This can be a tricky problem. If the transformation procedures are controlled strictly by input cues, they will have to look through much information that may eventually turn out to be irrelevant to the problem at hand, and which may even provide cues which are misleading with respect to the expert model, i.e., cause it to go off on a tangent. If the procedures are controlled strictly by abstract model features, they will be in the position of having to work through an entire problem-solving model which is too abstract and too general for any specific problem, again involving not only wasted work, but also difficulty in establishing the ramifications of individual problem specific features on the modelling effort as a whole. The approach taken here is to control the mapping procedures with features of the abstract model version of the particular problem which the system builds during each problem-solving effort. That is, as abstract model versions are made for subparts of the problem, special features are extracted from them and placed in the special three part data structure (known as the *theme*, *trend*, *edge* mechanism) which is used to guide the transformation procedures in building other parts. As more and more of the problem description is modelled, the information available to the transformation procedures gets better and better--more and more tailored to the specific needs of the expert's model for that problem. For example, suppose that during an earlier part of the modelling effort on the labor sector of the particular problem under consideration, it were discovered that the abstract model concept (PEOPLE (EMPLOYED)) should be modelled as three separate groups of employees (because of the structure of that particular firm). This information, i.e., the abstract model version of the three groups of employees in the firm, would be retained in the *trend* (the data structure for storing features which affect a closely interacting group of pieces) of the labor sector to indicate that the transformation procedures must take this feature into account when they work on pieces of the model belonging to the closely interacting labor sector group. It would thus be available to the procedures (in fact, it would force them to take that information into account) when it came time to create the abstract model version of the firm's hiring policy. It could have important implications on the way the construct (*) should be transformed by

the program; e.g., perhaps only one of the three groups of (PEOPLE (EMPLOYED)) are hired to control the backlog.

The process of constructing the mapping between the abstract concept of hiring shown in (*) and the features of the hiring policy in the particular firm under consideration shown in (**) is what I mean by reformulation. Incidentally, the result of the reformulation effort for this problem description (shown in figure 3)--involving, of course, other parts of the problem description besides (**)--is

```
((FIRM) (HIRE))
    (RATIO
     (DISCREPANCY
        (SUM (RATIO (((CUSTOMER) (ORDER)) (SMOOTHED))
                    (NUMBER-OF (LINE) 1000.))
             (RATIO
              (RATIO
                 (DIFFERENCE
                    (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                    ((NUMBER-OF (LINE) 1000000.)
                             MODIFICATION (DESIRED))
                 (NUMBER-OF (DAY) 10.))
              (NUMBER-OF (LINE) 1000.)))
        (SUM (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
             (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
             (NUMBER-OF (PEOPLE (LAID-OFF (WORKING))) NIL)))
     (NUMBER-OF (DAY) 5.))).
```

(The complete meaning and derivation of this hiring policy will be explained later in the thesis.) The rest of this chapter examines the particular problems involved with setting up an expert system in the way I have suggested here, explores the concept of an "abstract model" more thoroughly, and continues the example of reformulation begun here, describing how the transformation process is used to construct the mapping shown above. The next chapter goes into more detail about how this approach relates to other ways of building knowledge-based systems.

Section 1 The problem defined

The focus of this research is the study of a particular way of building expert systems and the construction of a working consulting program to show the effectiveness of the ideas presented. Although the scope of the program is necessarily limited (because the problems are so large), it fully demonstrates the implementation methodology that is developed here. Of course, part of the responsibility of the thesis is to show where the program's limitations come from, and how the general methodology described in the thesis can be applied to building other expert systems. But first I must establish the framework for this research.

I will begin with the question of what it takes to be an "expert". One thing that comes to mind immediately is that an expert must have extensive knowledge (gained through study, experience, or whatever) in his area of expertise. Of course, we also know that just "having" the knowledge is not enough. The expert is one who has *organized his expertise* in such a way that he can solve the problems that are posed in his particular area. My thesis concentrates on the problem of organizing expertise.

Now there are many different kinds of experts and thus many requirements for the organization of expertise. The "trivia expert" must organize his knowledge so that he can answer questions like "What was Dr. Zarkov's first name in the Buck Rogers series?". The doctor must organize his knowledge so that he can quickly diagnose diseases as described by his patient or by experimental data. My thesis represents the investigation of a particular problem in the organization of expertise--the reformulation approach described earlier.

This way of applying expertise is most effective in domains in which the expertise can be expressed as a well thought-out, self-consistent body of knowledge which acts as an environment *in which to do special reasoning*. The basic assumption is that people don't know much about how to reason in terms of most of what we know. They know a lot of relatively disconnected facts, and a few deductions relating them. However, there are domains for which specialized problem-solving models have been worked out. These models contain enough rules, consistencies, reasoning mechanisms, or whatever to make them capable of supporting much deeper problem-solving than is possible in the

unmodelled domains. I will call any domain with such an abstract model a "reformulation domain".

I think that experts who work in these reformulation domains do not represent their expertise in terms of and interweave it with the rest of their knowledge. In order to understand their expertise in terms of their normal representation, they would have to break it down and figure out what it implies in terms of all of their other concepts. But this would mean losing the special structure and coherence which is often the fundamental contribution of the model. Furthermore, the specialized deductive knowledge which is part of the model would be hard to recognize and to utilize if it were completely blended into with the quotidian details of the world at large.

Instead, reformulation experts maintain their expertise in a form which can be "reasoned about" easily--i.e., abstract, canonical, and free from non-essential detail. Instead of going into reweaving, they spend their effort on developing good ways to transfer problem descriptions from their "usual" representations to their special ones.

Note carefully that this is very different from simply *restating* the problem in terms of a modelling formalism. If the expert just formalizes the problem, i.e., translates it into the language of his abstract model, he will have gained little. True reformulation almost always involves a process of simplification and abstraction: the expert drives toward the conceptual economy (and thus understandabiltiy) of his pre-developed model. Let me make this distinction clear in terms of the problem area I am going to use in the thesis.

Consider the case of the business consultant who uses simulation modelling in order to solve problems. He deals with systems that are too complex to be understood on their own terms and for which satisfactory closed-form mathematical models are theoretically or practically unattainable. His job is to put the system into an understandable form so that its behavior can be altered or predicted.

The simple restatement of the problem in simulation terms is only a rather small step toward understanding the problem. It is relatively easy for an expert well versed in simulation techniques to go in and simply "simulate" the system. That is, represent (copying as closely as possible) the activities of the real system in terms of a simulation

model[1]. But he has done little to really advance his understanding of the system. The model is likely to be almost as complex as the real system itself--after all, it is just a translation. Simplifications may be made because certain parts of the firm are too hard to model, not necessarily because the simpler form maps into an abstract concept that the expert uses for doing deep reasoning. The simulator has gained something in that the simulation model is presumably easier to change, test, and explore. However, he does not *understand* the model very much better than the real system. He has simply restated reality in formal terms.

Contrast this to the case of the reformulation consultant who uses simulation as a technique. He has in mind a model of how certain business situations work. The nature of his abstract model is such that it is stated as a process--i.e., in simulation terms. His problem-solving technique is to see how his model applies to the particular real system he is confronted with. In order to do this, he will try to represent the real situation in terms of his model. He must strike a balance between adequately representing the real system and making it simple and understandable (i.e., in terms of his model). That is, he must capture the essence[2] of the real situation that makes it amenable to his model. The consultant has a tremendous stake in being able to do this; if he is unable, if he cannot see how to apply his abstract model to the problem, he will not be very much better off than anyone else: his expertise is in terms of the model. Thus, the reformulation expert starts with a model and drives toward representing the real system in terms of that model. He does not just start applying a modelling formalism and see what he ends up with.

To choose an example of reformulation which may be familiar to many people, consider the problem of applying Newtonian physics to high school physics problems. The student has a Newtonian model of how forces interact with masses. He views (i.e., models the characteristics of) certain problems in terms of this model. If he is asked to, say, understand the workings of a machine, his method is to describe the mechanism in terms of forces and masses: in terms of the model he can work with. Now the student knows that "force" and "mass" are simply abstract concepts. He also knows that he will have to simplify the machine in order to figure out how the concepts of force and mass apply to it. Nonetheless, he knows that once he has done so, he will have a very deep understanding

[1] In fact, a program has been written to do this for simple queuing problems--see [Heidorn].

[2] I will give this notion of "essence" a rather specific meaning in section III-1.

of the new modelled system. He can solve very sophisticated problems which he could never have handled in non-model terms. He can come up with insights into the nature of the machine that were completely hidden from him before. The point is that the student comes in with a model he (or at least someone) understands deeply. He devotes much of his overall problem-solving energy to casting the problem in terms of that model so that he can bring this deep understanding to bear. If his model has really "captured the essence" of the real system, he will be able to apply deep understanding to the same situation which was previously completely intractable. Let me now continue with a less familiar but more germane example of reformulation.

Suppose a business consultant has an abstract model which contains a few models that deal with how problems of labor fluctuations arise, and how such problems can be fixed. A common complaint of managers is that fluctuations in their workload cause them to hire and layoff workers much more often than they would like (high turnover is expensive, unfair, etc.). The cost of this oscillation in the level of the workforce can be very high in terms of direct turnover costs, poor use of equipment, overtime, managerial time, etc. Clearly, productivity and profit could be improved if these fluctuations were levelled off or at least attenuated. Managers often give as the reason for these expensive fluctuations similar fluctuations in the incoming workload (highly variable requests for work, changing customer base, fads, etc.). The way in which a manager might describe a typical example of workforce fluctuation problems is represented in condensed "case" form (from [Jarmain]) in figure 3 below.

Dominion Typesetters, Inc., is a small company engaged in typesetting. In the past, this industry has been very competitive. Customers requiring typesetting usually investigate both price and estimated delivery time before placing an order.

Labor is by far the most important cost component in this business. Consequently, Dominion has found it necessary to make most efficient use of labor in order to keep prices at a competitive level. The practice with Dominion as well as its competitors has been to maintain a backlog of orders so that day-to-day random fluctuations in incoming orders will not leave part of the work force idle at certain times. Even with the order backlog as protection against day-to-day fluctuations in incoming orders, they have often found it necessary to hire or lay off typesetters. Hiring has taken place when the order backlog became large and sales were being lost to competitors. Layoff has taken place when the order backlog became so small that day-to-day fluctuations in incoming orders threatened to leave some of the work force idle. If even a small portion of the work force is idle, Dominion will incur losses under the present highly competitive prices.

At the moment, relatively little is known about the characteristics of the customers in this industry. Dominion has found that its sales have tended to decrease when its order backlog has become much longer than the 2-week period which is average for the industry. Similarly, its sales have seemed to increase when the backlog has fallen much below this 2-week figure. In the long run, both Dominion's sales and industry sales have remained relatively constant. However, there have been bad periods and good periods for which there does not seem to be much explanation.

Dominion's work force is composed primarily of union typesetters. Each of these typesetters usually produces at the rate of 1000 lines, or 1 kiloline per day. If additional typesetters are required, it usually takes 5 days, on the average, from the time the decision to hire is made until new men are actually at work in their jobs. When layoff is necessary, the average notice period is about 5 days. In the past, there has always been a substantial number of typesetters looking for employment. However, the management at Dominion has refrained from indiscriminate hiring and layoff for the fear that the union would press for higher wage rates. This would place the company in a difficult position with respect to some of its competitors whose typesetters belong to a different union.

At the moment, Dominion has 100 typesetters in its employ, and the company is producing at the rate of 100 kilolines per day. This figure is approximately equal to Dominion's average daily order rate over the last 8 weeks. There is an unfilled-order backlog of 1000 kilolines representing 10 days of production. This backlog represents approximately a 2-week delay in meeting new customer orders on the basis of a 5-day work week. The delay is about average for the industry. No decision to hire or lay off has been made during the past several weeks. Dominion's prices are approximately equal to those of its competitors, and top management has decided to maintain the present price level over the next few months at least.

Figure 3. Manager's view of the problem

I think that we can take this as a fairly reasonable representation of the way in which the manager views the problem. What does the consultant think about it? I mentioned above a consultant's "abstract model" which deals with this very problem. To expand on this, consider the following "workforce need" model (after [Forrester]). Although input workload fluctuations are usually the root cause of workforce need fluctuations, their bad effects are often exacerbated by policies *within* the system (i.e. directly controllable by the firm). This is often true to the degree that if it were not for the adverse effect of improper policies, input workload fluctuations would cause only minor and very acceptable workforce fluctuations. These "improper policies" often arise because of poor understanding of the interaction between sales, production, inventory (or backlog), and hiring and firing policy.

There is often a "feedback" relationship between these sectors:

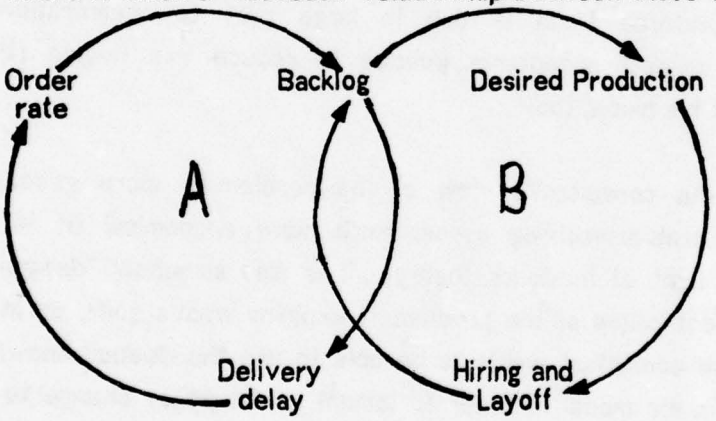


Figure 4. Feedback loops surmised by the consultant

That is, for example (following loop "A"), when the customer order rate reaches a certain level, the company must start backlogging orders. Customers whose orders have been backlogged experience a longer than usual delay in having the orders filled. This in turn often causes them to place their orders elsewhere if there is a competitor which can deliver the goods with less delay. Finally, this causes the backlog to be reduced as less incoming orders need to be filled. Thus, these activities form a "feedback loop" and affect each other in ways that are very non-intuitive to someone not versed in feedback theory.

There is also often a feedback loop ("B") relating backlog, production, and personnel needs. Note that the two loops interlock, creating interactions which can seem terribly complex to the manager.

In the case of Dominion, the consultant might propose that decisions made in the face of this complexity (or poor understanding of the feedback relationships, from the consultant's point of view) have lead to a poor choice of policies at several points within the structure of the firm. For example, policies which cause "overshooting" of the perceived need for change in productivity (e.g., "more productivity to decrease a backlog, less when the backlog reaches an acceptable level") accompanied by the time delays usual in such a system (time to train and layoff, etc.) have probably lead to the chronic "amplification" of the workload fluctuation which has been causing the problems in the firm. Thus, the consultant might suggest, based on this model, that the prolonged fluctuation ("ringing") of the workforce level is due in large part to organizational policies. Furthermore, he can suggest acceptable policies to reduce this ringing (these policy suggestions are part of his model too)[1].

Note that the consultant's view of the problem is more general than the manager's, and, in a problem-solving sense, much more economical (it is simple and understandable in the light of feedback theory). It is also somehow "deeper" in that it seems to get at the root cause of the problem: it explains what's going on in the system and what's wrong. The consultant wants to be able to use the deeper knowledge of the problem represented in his model in order to foment useful policy change to correct the system's disorders. In order to do this, the consultant must apply this theory to the problem at hand. The technique of applying such a model to a problem description at the level of figure 3 is a realistic example of reformulation. The objective of this thesis is to create an implementation methodology for computer programs which provide reformulation-based expertise. In Chapter VIII I will show how a program built with this methodology would actually perform the reformulation task required to apply the abstract model discussed here to the problem of figure 3.

Right now I will emphasize some of the characteristics of this problem that make it not only difficult, but also rather different from other expertise-application problems.

[1] The workforce need model is discussed further in V-1.3.

Section 2 Characteristics of the problem

In this section I will consider the special "organization of expertise" problems which the reformulation process tries to solve. In order to get a better grasp on these problems, the discussion will be in the context of an actual example of the program's reformulation effort in the case of figure 3. This will be used to define the particular demands which are made on the consultant's expertise organization in a reformulation environment. These "demands" will be used in Chapters II and III as criteria to be met by an implementation of the reformulation process.

It is clear that in order to figure out how to apply the workforce need theory to Dominion, the program must be able to understand Dominion's hiring policy in terms of its model. The program uses a cause-effect flow model of the business domain. Its model of the way workforce is controlled by hiring policy can be represented as follows (a fuller version of figure 2):

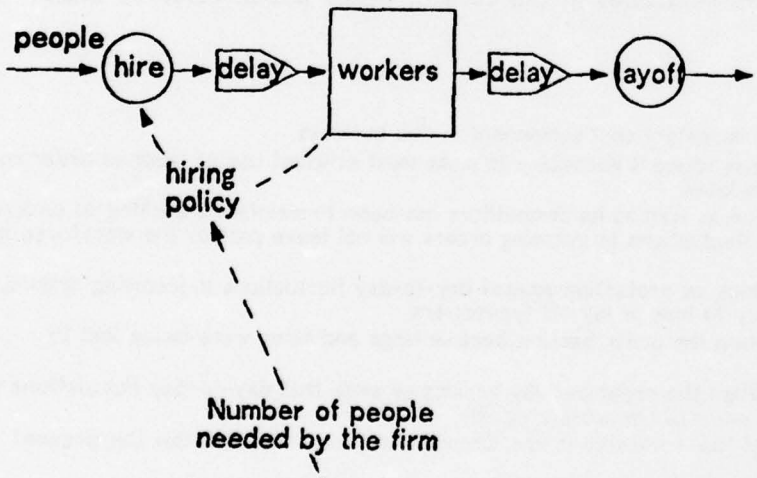


Figure 5. Abstract model of hiring and workforce

That is, people flow through the firm subject to its actions and policies. They

flow into the firm by the action of the hiring activity. Once hired, and after some delay, they are considered to be workers of the firm. The flow of people out of the firm is by action of the layoff activity, again subject to a delay. In addition, the hiring activity is controlled by a hiring policy which tells the hiring activity to what extent it should allow people to flow into the firm. The hiring policy considers two factors in its decision: the number of workers already in the firm, and the number needed by the firm at that particular time. In fact, as we saw earlier, the expert views the hiring policy as a feedback control mechanism--hiring affects the number of workers, which affects the hiring policy, which in turn affects hiring. The policy is involved in keeping the "current number of workers", the feedback variable, as close as possible to the "number of workers needed by the firm at any given time", a variable determined by parts of the model not shown in figure 5. For the expert, then, the relationship between hiring and workforce is that of flow control via a classical feedback control mechanism which attempts to reduce the difference between an observed variable and an exogenously provided goal.

Now let's see what the manager has to say about hiring and workforce. Figure 6 below shows all of the sentences in the case of figure 3 that refer to either hiring or workforce:

1 Labor is by far the most important cost somponent in this business.
2 Consequently, Dominion has found it necessary to make most efficient use of labor in order to keep prices at a competitive level.
3 The practice with Dominion as well as its competitors has been to maintain a backlog of orders so that day-to-day random fluctuations in incoming orders will not leave part of the workforce idle at certain times.
4 Even with the order backlog as protection against day-to-day fluctuations in incoming orders, they have found it necessary to hire or lay off typesetters.
5 Hiring has taken place when the order backlog became large and sales were being lost to competitors.
6 Layoff has taken place when the order backlog became so small that day-to-day fluctuations in orders threatened to leave some of the workforce idle.
7 If even a small portion of the workforce is idle, Dominion will incur losses under the present highly competitive prices.
8 Dominion's work force is composed primarily of union typesetters.
9 If additional typesetters are required, it usually takes 5 days, on the average, from the time the decision to hire is made until new men are actually at work in their jobs.
10 However, the management of Dominion has refrained from indiscriminate hiring and layoff for the fear that the union would press for higher wage rates.
11 No decision to hire or lay off has been made during the past several weeks.

Figure 6. The manager's description of hiring and workforce

The task of the program is to see how its model of figure 5 applies to the information in figure 6. Specifically, consider the following:

The expert's language for describing a particular concept is different from that of the manager. The expert's model of figure 5 is designed to be applicable to a wide variety of situations like that in figure 6. In order for this to be possible, the expert's concepts must be more abstract than those of the manager. Part of the reformulation task is to recognize the correspondence between the manager's concept and the expert's model of that concept. I call this the *abstraction* problem.

In fact, the expert's choice of concepts, i.e., coherent describable entities, is different from that of the manager. The order in which parts of the manager's situation are described, their interconnection, and their relative importance differ greatly from the needs of the expert model. The problem of reformulation is therefore not a simple matter of associating a more abstract concept to a problem-specific one. Significant rearrangement and restructuring of either the expert's model or the manager's description is necessary before the expert can see how his model applies to the problem. I call these non-abstraction parts of the concept-matching task the *focussing* problem.

Some of the information in figure 6 is irrelevant to the expert model of figure 5. The program must be able to plow through the problem description and ferret out what it needs to know. On the other hand, some information needed by the program to complete its model is missing from the manager description. The program must know enough to ask the user when it needs to know something. The task of gathering the information needed for the reformulation effort will be called *selection*.

In any reasonable-sized reformulation effort, it is the case that some parts of the manager description must be modelled in great detail, while other parts can be handled in a more cursory fashion. Furthermore, even a small piece of the abstract model such as the little flow in figure 5 may be used in more or less detailed ways by the program, even within the

same modelling effort. We will see in the program's handling of the Dominion case (Chapter VIII) that the model in figure 5 is first used in a very non-detailed way to see whether the flow of workforce through Dominion is at all the way the workforce need theory expects it to be, and then in a more specific way to look for faults in the way Dominion actually does its hiring. The point is that in one case the manager's information is too detailed: the program only needs to garner a few major facts, not to check through all the detailed implications of what he has said. In the other case, it is these very implications that he must know in order to figure out the exact nature of Dominion's difficulty. This problem, which is really a matter of varying the needs of the selection task, will be referred to as the *level of detail* problem.

Finally, the fact that the expert's model of hiring is designed to be applied to a wide variety of problems means that no problem will require the model in its entirety. Even the little model of figure 5 contains more knowledge than is necessary for its application to the description of figure 6. The program must apply only that part of the model which is necessary, so that it it will not waste time trying to deal with impossible eventualities or asking the user questions he can't answer. I call this *model tailoring.*

These fundamental problems keep the process of reformulating the naive problem description into abstract model form from being any kind of simple matching procedure: the manager's description of the organization will not correspond directly to *anything* in the consultant's model. In fact, the "most natural way of describing the problem" may lead perversely away from the abstract scheme of the consultant. To counteract this diversionary trend, the consultant usually does reformulation by keeping his model constantly in mind and getting the manager to tell him about pieces of the problem which are covered by the piece of model the consultant is currently working on[1].

Of course, getting the manager to talk about the problem in this way is by no means straightforward. The consultant must use the manager's terminology in asking

[1]On the other hand, the consultant is not allowed to talk the manager into a problem just because the consulting model is good at solving it--that is, I will consider this to be unfair, even though it is a fairly common practice.

questions, but must use the model's terminology in keeping track of the answers. Furthermore, this terminology problem is just a reflection of the larger issue: the consultant must use the manager's conceptual scheme when dealing with the manager, and the model's conceptual scheme when dealing with the model. For example, when asking questions or explaining things to the manager, the consultant must arrange things so that they are amenable to the manager's way of thinking. If the consultant is talking about the decision rules which influence the re-ordering of inventory, he should not suddenly change to a discussion of how to damp fluctuations in the retail sector, even though this is what the conceptual structure of the abstract model dictates. Similarly, the consultant cannot use the "natural" conceptual structure of the manager to relate things in the model. And, since there is no straightforward mapping between the conceptual schemes, some sort of intelligent processing must do the reformulation necessary to bring them together.

An expert program which works by reformulation must also do this "intelligent processing". This thesis is devoted to the discussion of a scheme for implementing the reformulation process in a computer program. I will preview this scheme in section 4 of this chapter. Before getting into this, though, it's probably a good idea to convince yourself that the scheme is worth looking at. I provide help for this in the next section.

Section 3 The importance of the problem

As I said earlier, reformulation is both a model of expertise (as described in the previous sections) and a model of how to implement expertise in computer programs (as described in section 4). In this section I will try to bring these two aspects together.

The basic reason for studying reformulation is that it is a real and useful aspect of expertise: the reformulation process is a good description of the way in which some experts solve problems. Evidence for this is easy to find; versions of the reformulation paradigm have been articulated by "practicing experts" (e.g., consultants, see [Gorry] and [Forrester]; anthropologists, see [Geertz]; etc.) as models of what they do, and by "studiers of expertise" (e.g., [Kuhn]) as models of what certain experts (scientists, in the case of Kuhn) do. Thus, any implementation of the reformulation process could be seen as a program which could provide (some well-defined part of) the problem-solving expertise of, say, a consultant.

Although the thrust of the thesis is in this direction, i.e., providing a totally reformulation-based expert system, reformulation itself has much wider applicability as a *part* of other kinds of expertise. Many experts (and non-experts) who use other problem-solving methods find that reformulation is useful at a certain stage of the process, or for a certain portion of their problems. The idea of understanding things in terms of a made-up or learned model is very fundamental. While not everyone has the thoroughgoing model of a scientist or reformulation-type consultant, almost everyone does sometimes find that making up and using (or just using) an abstract model is a good way to solve a problem.

In fact, the process of making up, using, and progressively improving models has been suggested as a representation of the learning process itself [Papert]. There is also evidence that managers (e.g., see [Pounds]), mechanics, doctors, lawyers, etc. use (perhaps very simple) abstract models to explain certain things and to solve certain problems. For example, although lawyers work almost entirely by case analysis and "recognition" (see section II-3), they may have a few abstract models of "how the law works" in certain isolated areas (e.g., patent law). All of these people must use reformulation at one time or another to apply their models to their problems. Thus, reformulation is used over a wide range of expert and non-expert problem-solving processes.

But reformulation is more than a theory of organizing knowledge. It also turns out that some important aspects of "being expert" can be very naturally implemented in a reformulation environment. In order to get at these "important aspects", let's consider a few things we can (hopefully) agree are characteristic of *all* experts, not just the reformulation variety we have seen in the previous two sections.

First, then,

Experts know a lot of "things" about their domain of expertise (e.g., "blood in the urine can be due to a variety of causes").

Experts have tricks for applying these "things" to problems in their domains (e.g., "a good way to get a rough idea of how much blood is getting into the urine is to have the patient compare the color of the urine to a well known shade of brown (e.g., coffee)").

All expert problem-solving programs handle expert knowledge and expert tricks in some way (though there is little agreement on what is a "good" way, as we will see). However, if we extend these capabilities a little, we already begin to move beyond the state of the art...

Experts who deal with reasonably large and complex domains have many tricks and much "possibly applicable" information; only a small part of what they know (and only at a certain level of detail) is needed to solve any particular problem. Decisions about "how much to use of what when" can usually be made only *during* the problem-solving effort. Much of the expert's "expertise" is involved with controlling and structuring his problem-solving effort in order to "tailor" it to the specific problem at hand. That is, all experts have level of detail and model tailoring problems which must be solved by their expertise application process. (E.g., "if a patient previously treated for a broken nose complains of coffee-colored urine, it's worth checking to see whether the blood in the urine is due to direct trauma to the kidneys (the patient might be involved in contact sports, etc.))

All expert programs have some capability for sifting through their knowledge and applying it only as necessary. However, as we will see in the next chapter, none of the current methods are able to make use of both local and global "tailoring" knowledge in a manageable way--something that I consider extremely important for problem-solving in a big domain. Furthermore, current methods do not "tailor" so much as choose between alternatives or expand "macros". This makes it difficult to handle things like level of detail considerations.

One more observation on experts in general:

Experts are frequently in the situation of dealing with information from knowledgeable but unenlightened sources. The result is that though experts must use the problem description to guide their problem-solving effort, they must also plow through irrelevant information (perhaps thought

relevant by the informant) in order to home in on actual problems and important facts. This is of course the selection problem. (E.g., "if the broken nose turns out to have been due to a tiff with a passing stranger, it might do to look for a different cause".)

With this consideration we leave current expert programs behind. This problem simply has not been important to the limited scope expert programs developed in the past. Existing programs either take input directly in the form of what are essentially "goals" for the problem-solver (e.g., HACKER [Sussman], MACSYMA [Moses], Mycroft [Goldstein]) or extract the needed description of the problem through rigidly controlled questions and answers (e.g., MYCIN [Shortliffe], PROCTOR [Bosyj]). There is never the problem of weeding out extraneous data or even of delving into the description to see what's really there. This is clearly different from any real problem.

My claim is that the reformulation approach is worth studying because it leads to an implementation which is especially good at providing these last two features of general expertise--the "real world" features of level of detail and selection. This comes about from the particular way in which the abstract model is applied to problems--a process which I'll describe in the next section.

Reformulation is interesting, then, first, because it is a useful model for a kind of expert problem-solving, and second, because its implementation philosophy is well adapted to attacking the problems of working in real world domains.

Section 4 Preview of the implementation methodology

In this section I will briefly describe the way in which the implementation methodology developed in this thesis attempts to solve the problems of reformulation listed in section I-2. I will illustrate the methodology with examples from the program's effort to apply the model of figure 5 to the problem description of figure 6. The implementation methodology is discussed in greater detail in Chapters V, VI, and VII. A full account of the program's reformulation effort in the Dominion case is given in Chapter VIII. Here I am just going to sketch the overall mechanism to establish a framework in which to read the following chapters.

First of all, then, the flow part of the model of figure 5 is represented in the program as

```
(DETERMINED-BY (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
    (INFLUX (DELAY1 (ACT-ON (PEOPLE)
                            ((FIRM) (HIRE)))))
    (OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                             ((FIRM) (LAYOFF))))))).
```

The hiring policy part of the model, shown earlier, is

```
(SMOOTH (DISCREPANCY
        ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED))
        (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))).
```

This notation is described fully in Chapter V. I am using it here because it is a key aspect of the way in which knowledge is organized in the program.

In particular, I wish at this point to distinguish the two kinds of elements in the representation that I mentioned briefly before. First, there are the ones like (PEOPLE), (PEOPLE (EMPLOYED (WORKING))) (i.e., workers), and ((FIRM) (HIRE)) (i.e., the hiring activity of the firm). These are the abstract concepts of the expert's model referred to earlier. They are to be associated with pieces of the manager's description as part of the modelling effort. That is, the expert's concept of (HIRE) must be identified with the manager's description of the firm's hiring activity for the model to be applied. Then there are the things like INFLUX, OUTFLUX, and DETERMINED-BY. These express structural relationships within the model, and are not expected to be associated with pieces of the problem description.

This distinction is important because of the knowledge attached to the different kinds of objects. The knowledge attached to the abstract concepts like (PEOPLE) and (HIRE) is in the form of simple database access routines which say what that concept can be associated with (i.e., matched to) in the problem description. These are used to solve the abstraction problem--the concept-to-concept association of a piece of the model to a

piece of the input description--and are called **abstracters**. For example, the abstracters attached to (PEOPLE (EMPLOYED (WORKING))) are capable of matching it with workers, employees, laborers, workforce, etc. When the program is told that Dominion workers are "typesetters", the abstracters are told to accept that too as a match for (PEOPLE (EMPLOYED (WORKING))).

Attached to the objects like DETERMINED-BY is *modelling knowledge*, which tells the program how to go about modelling the particular kind of structure they represent. In this case, the modelling knowledge attached to DETERMINED-BY says that since (NUMBER-OF (PEOPLE (EMPLOYED (WORKING)))) is determined by an INFLUX and an OUTFLUX (which represent flows into and out of something), the whole structure should be treated as a flow and modelled piece by piece, left to right. More important is the fact that within a flow like this, the abstract objects of the model have special relationships with each other. For example, anything that affects the flow of people through the (HIRE) activity is liable to affect the DELAY which is just "downstream" from it. This knowledge about flows (a full description is given in V-1.2) is also part of the modelling knowledge. Modelling knowledge is used by the program to solve the selection problem. When the program is trying to apply the model of figure 5 to the information of figure 6, it will concern itself only with those statements which tell it something that is important in the "flow" sense, that is, that tell it something needed by the modelling knowledge attached to the DETERMINED-BY flow. It will essentially go through the flow step by step, modelling each activity to the extent required (as dictated by the level of detail restriction discussed below), and using what it has learned about the previous activity to help it model the next. Information that does not refer to how activities relate to each other in the flow, i.e., in this case, how people flow from one activity to the next, is ignored. For example, sentences 1, 2, 8, etc. of figure 6 say nothing about the interrelationship of activities defined in the flow, so they are ignored in this part of the modelling effort. On the other hand, sentence 9 does give information about the relationship of the hiring activity to the workers of the firm (and, inferentially, to the delay between them), and would be used when the flow is modelled. Furthermore, if, in trying to model one of the activities of the flow, needed information was found to be missing (and could not be deduced from the rules of flows), the user would be asked for the needed information. Therefore, the program does selection by "sticking to its model"--using the modelling knowledge attached to the objects like DETERMINED-BY to decide what information it needs.

The above two kinds of knowledge, abstracters and modelling knowledge, form the only data that is attached to the expert model of figure 5. Note something very important: although the knowledge attached directly to the objects of the abstract model contains everything necessary for their local use in the modelling effort, it contains no information about when and how that knowledge should be used. In particular, it does not say how to solve the focussing, level of detail, or model tailoring problems mentioned earlier. It says nothing about how the hiring policy structure shown above is to be associated with the manager's description of hiring in sentence 5 (the structure as represented is certainly not an "abstraction" of that sentence--there is no concept-to-concept match). It does not say how the flow should be modelled once as a general check and then again at a more detailed level, as described above. It does not say how the rest of the model should be tailored when it is discovered, for example, that the concept (PEOPLE (EMPLOYED)) really consists of three separate groups which must be accounted for. All of these problems are handled by the modelling mechanism which sets up and associates the pieces of the abstract model. This is a very basic principle of the implementation methodology discussed in this thesis: all the local knowledge about a particular piece of the abstract model is attached directly to that piece, and only that local knowledge--global control structure considerations, the problem of knowing how to use what when, is handled by the modelling mechanism.

The modelling mechanism works by setting up a "chunk" of the abstract model (the DETERMINED-BY structure for the flow part of figure 5 shown above is such a chunk, the SMOOTH structure representation of the hiring policy is another), associating it with a piece of the problem description, and placing the resulting association into the **modelled system** (the model of the program so far):

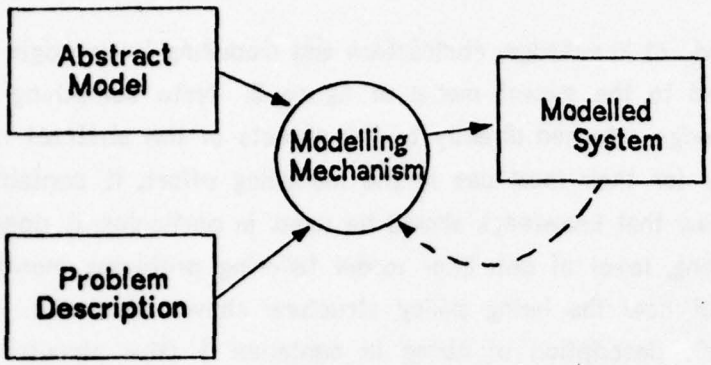


Figure 7. Schematic of the modelling mechanism

Note the similarity to the basic model of reformulation shown in figure 1. The whole modelling process is described in Chapter III. Right now the only thing that will concern us is that little dotted line in figure 7.

That dotted line represents the second basic principle of the implementation methodology discussed in this thesis: the program's knowledge of how to use what chunk when is controlled via feedback through the modelled system; the modelling mechanism looks at what it has done to see what it should do next. Unfortunately, the modelled system itself is hard to look at since it contains all of the details of the program's model of the problem descripition. Therefore, the program selects certain features of each freshly modelled chunk as it puts it into the modelled system. These features are stored separately and used to help the modelling mechanism with focussing, level of detail handling, and model tailoring. As the modelling effort proceeds and more and more information is learned about the specific problem at hand, the modelled feature data available to the modelling mechanism gets better and better.

The selected features are kept in special data structures known as **themes**, **trends**, and **edges**. Themes express the overall goals of the problem-solving process. Trends are used to hold information which affects the modelling of a group of chunks. Edges are the constraints of one piece on another, the "ports" of each piece. The basic

operation of the program is therefore really: select a chunk; use the information in the theme, trend, edge mechanism to see whether the chunk should be handled at a different level of detail (i.e., have its use restricted because of the needs of the overall problem-solving effort), focussed (i.e., structurally changed so that it can be matched to a piece of the problem description), or tailored (i.e., structurally changed to reflect known properties of the model so far) to set up the chunk for association; associate it; select features of the associated chunk and place them in the theme, trend, or edge; place the associated chunk in the modelled system. In Chapters III and VII we will see how features are selected for the theme, trend, edge mechanism, and how each of the parts of the mechanism is used. Here I will show some examples of how the trend is used, since it is the most interesting of the three parts, and since it has the most bearing on the problems of level of detail, focussing, and (to a lesser extent) model tailoring.

I said earlier that the program first uses the model of figure 5 simply as a check to see whether the workforce need theory is applicable to Dominion. The program is essentially trying to determine, without going into too much detail, whether or not the flow part of figure 5 is a good model for what is going on in Dominion. If it isn't, the program can't use the workforce need theory. Thus, there is no point going into the details of any of the individual activities until it is known that the *whole* flow is applicable to Dominion. Note that this goes against the modelling knowledge attached to the flow which I gave earlier: a DETERMINED-BY flow expects to model each part of the flow in detail one by one, left to right--a fairly significant effort. It is the modelling mechanism which must tell DETERMINED-BY not to go ahead and do this, since this change of modelling strategy is a level of detail decision, and thus clearly a global control structure issue.

Level of detail restrictions are enforced via trend information. When the decision is made to verify figure 5 as a model for Dominion's workforce utilization, the objective of the modelling effort is entered into the trend. In this case, it is to see how hiring and layoff are related in Dominion. Also at this time, level of detail restrictions are placed in the trend, based on this objective. They are in the form of the abstract entities which make up the objective, in this case (HIRE) and (LAYOFF). When the DETERMINED-BY chunk is set up, the modelling mechanism notes the presence of (HIRE) and (LAYOFF) in the trend, meaning that nothing at a lower level of detail than (HIRE) and (LAYOFF) should be modelled in this effort. We'll see exactly what a "lower level" than (HIRE) means in Chapter VII. For now I will say that it includes things like the hiring policy chunk which

defines (HIRE). Therefore, when the modelling knowledge of the DETERMINED-BY construct tries to model (HIRE) by setting up the hiring policy chunk, it is restricted from doing so, and instead (HIRE) is simply modelled as (HIRE)--i.e, verified. This can be satisfactorily done by looking at the information in the second clause of sentence 4 of figure 6 or the first clause of sentence 5, which just says that people are indeed being hired. With this quick check it can verify the model of

```
(ACT-ON (PEOPLE)
        ((FIRM) (HIRE))
```

and go on with the rest of the flow.

This is in marked contrast to what happens when this same DETERMINED-BY chunk is modelled in a more detailed way, as we'll see in this example of focussing. Let us assume that the workforce flow has been verified as above and that the program now wants to model each of the activities of the flow in order to find out what's really going on. Since there is no level of detail restriction to the contrary, this time the modelling knowledge attached to DETERMINED-BY sets up the hiring policy chunk

```
(SMOOTH (DISCREPANCY
        ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED))
        (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)))
```

for association when it comes to model (HIRE). The only piece of the problem description that says when people are hired by Dominion, i.e., gives some notion of the hiring policy, is the first clause of sentence 5 which was used for verification before. It relates hiring to order backlog.

Now this sentence doesn't seem very useful. The expert's notion of a hiring policy, as expressed by the above chunk, is that it is a workforce discrepancy reducing mechanism. The manager's notion is that it is something to do when the order backlog gets large. This is what I meant when I said earlier that the manager's choice of concepts and their interrelationship may differ radically from that of the expert. When the hiring policy

chunk is set up, its abstracters can see no relationship between the description of hiring in sentence 5 and the chunk as it is set up. The chunk must therefore be focussed, i.e., structurally changed to be made closer to the piece of problem description.

This focussing process is again controlled by information in the trend. One of the major focussing techniques of the program is to choose an alternate statement (called an **elaboration**) of a chunk--if there is one--when the abstracters fail on the initially proposed chunk. Elaborations model alternate policy choices, different definitions of concepts, different ways of calculating the same variable, etc. Again, elaborations are part of the local knowledge of the chunk. But it is the modelling mechanism which decides which elaborations of the concept should be used for any particular focussing purpose. This is done by looking at the trend. As mentioned above, when a chunk is set up, the objective for modelling that chunk is put into the trend. In this case, the objective is simply to model (HIRE), so the trend objective is (HIRE) itself. This means that any elaboration of any (HIRE) can be used for focussing (as long as it satisfies the other trend restrictions, like level of detail). If the objective was instead something like "model (HIRE) as an inventory controller", only elaborations of (HIRE) which dealt with inventory control could be used. At any rate, in this case the modelling mechanism is free to choose any elaboration of (HIRE). Therefore, when the modelling effort gets stuck trying to model the piece ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL) MODIFICATION (DESIRED)) (i.e., the abstracters of this piece can't find anything to match with it), the modelling mechanism finds an elaboration of the piece which deals with backlog--the one we saw earlier in this chapter (all of the elaborations of (HIRE) are shown in Chapter V). The level of detail of the new elaboration is okay, and the hiring policy chunk becomes

```
(SMOOTH (DISCREPANCY
        (SUM (RATIO (SMOOTH ((CUSTOMER) (ORDER)))
                    (PRODUCTIVITY))
             (RATIO (SMOOTH (DIFFERENCE
                             (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                             ((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                                 MODIFICATION (DESIRED))))
                    (PRODUCTIVITY)))
        (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))).
```

Clearly, more than just the information in sentence 5 is going to be necessary to associate this chunk. Also, we will see in Chapter VIII that more focussing must be done on it before matching can take place. However, most of the focussing efforts by the program are of the kind described here. A full account of focussing is given in Chapter III.

Let's stick with this same chunk for an example of model tailoring. When the DETERMINED-BY chunk is modelled at a high level of detail (i.e. "verified"), it is discovered that in the Dominion case, the abstract concept (PEOPLE (EMPLOYED)) must really be represented by three separate abstract concepts. This comes about because the program knows (via knowledge attached to (HIRE)) that people are (EMPLOYED) right after they are acted upon by (HIRE), and (via knowledge attached to (LAYOFF)) that they cease to be (EMPLOYED) after they are acted upon by (LAYOFF). This means, since "whatever flows in must flow out" (i.e., part of the flow knowledge), that all of the people in the flow between the hire and layoff activities in figure 5 must be (EMPLOYED). However, this consists of three groups: those between the hiring activity and the first delay (which turns out to be modelled by a "training" activity), those that are workers, and those between the worker box and the layoff activity (the delay between the worker box and layoff gets subsumed into layoff--another example of model tailoring that we will see in Chapter VIII). Thus, when the (PEOPLE (EMPLOYED)) concept is put into the modelled system, it will be represented as these three groups (trainees, workers, and people that have been laid off but haven't left yet). Now whenever the program goes to reexamine the flow of figure 5, and the objective of so doing is placed in the trend, the trend will not contain simply (PEOPLE (EMPLOYED)), but all three groups. In this way, the discovery is propagated throughout the modelling effort, and the model is more tailored to the problem situation at hand.

I have gone through this sketch in order to give the flavor of how knowledge is organized in the program, and how it is used to solve some of the problems mentioned earlier. This should help in comparing the approach taken here to those that are described in the next chapter. The details of how it all works are presented in overview fashion in Chapter III (to get the big picture), and then piece by piece in the following chapters. Chapter VIII then gives a complete account of what happens when these techniques are applied to the Dominion case of figure 3.

Chapter II
Showing That Many Men Have Many Minds

There has been a great deal of work put forth in computer science toward the development of "understanding systems" for a variety of applications (mostly natural language discourse situations). Some of these systems seem to be excellent first steps toward implementing various kinds of expert problem-solving mechanisms. However, I believe that none of them would provide an adequate basis for an implementation of the reformulation process. This section presents a critical analysis of some of the most relevant understanding systems developed (or discussed) so far from the point of view of their applicability for implementing reformulation. Before going on, I would like to make this critical viewpoint a bit clearer.

First of all, I will be considering only those problem-solving methodologies which have been successfully applied to "big" problem areas. I feel that the only valid candidates for technologies to implement reformulation are those which are designed to face the complexity of knowledge organization in a reasonable-sized application domain. Experience has shown that any system, no matter how great its theoretical potential, which cannot handle the necessary variety of representational and problem-solving-procedural issues is doomed to failure. Thus, I will exclude from consideration some of the most well-known problem-solving techniques (e.g., predicate calculus theorem provers, GPS-type schemes, etc.) at the outset.

Second, anyone who performs any sort of critical analysis of complex methodologies must deal with the fact that any methodology can "probably be expanded" to handle any given problem. In this section, I will try to make clear for each methodology certain basic philosophical or deeply-rooted implementational emphases which make it inapplicable for reformulation. That is, my argument will usually be of the form that it requires such a stretching or re-thinking effort to bend a given methodology into a shape that is appropriate for reformulation that the methodology should simply be considered inapplicable. I therefore attempt to eliminate each of the alternatives as an implausible, i.e., *unrealistic* technology for doing reformulation.

Third, it is quite clear that none of the techniques I will examine in this section were *intended* to do anything like reformulation. Indeed, most were never intended to be in any kind of expert system environment at all. Therefore, I try to consider in each case what would happen *if* I tried to apply the methodology to the reformulation problem. That is, my critical viewpoint is "*why can't I use an existing piece of work?*".

Finally, it is also obvious that this requisite "discussion of work to date" section could be book-length. Since some compromise is necessary, I will not scruple to compromise everything. I will divide the existing workers in the field of "implementing understanding" into four broad categories:

-1- primitivizers
-2- world-model-bassi
-3- recognitionists
-4- straight-shooters

and proceed to discuss them in the large.

Section 1 Primitivism

A number of schemes have been suggested for providing understanding systems to serve as "semantic interpreters" for *any* natural language discourse. They share the common philosophy that incoming information should be resolved into its constituent "primitive" concepts. Structures of these primitives are then used as the basis for further interpretation (inferences are made on the basis of them, larger concepts are handled in terms of them, etc.).

This underlying philosophy that reasoning should be done in terms of universal, low-level concepts brings together such vastly different methodologies as "semantic networks" [Quillian] and "conceptual depedency structures" [Schank]. These two differ

almost completely on the way in which primitives are to be derived, what are to be chosen as "primitives", etc.--but this is irrelevant here. I will try to show that it is the idea of primitivization itself that is inappropriate for reformulation.

There are two possible ways in which primitivizing could be useful for implementing reformulation:

-1- Universal primitives might be useful for translating the "natural concepts" of the problem description into the concepts of the expert's model.

-2- It might be possible to derive a highly specialized set of "primitives" of the expert's model, and primitivize *directly* into these.

The first point relates to the most straightforward method of applying a primitivizing technique to reformulation: incoming information is broken down into the "usual" set of primitives, and the expert, say a business consultant, uses structures of these primitives to fit the problem description into his problem-solving model. Thus, consider an input from the manager like "my stores depend on rapid delivery". This would be immediately broken down into the appropriate MTRANS's, PTRANS's, etc. [Schank] (or "store" would be *intersected* with "delivery" to find the right primitives [Quillian], or whatever), and then this new structure would be given to the consultant's model for interpretation. For example, in the Schankian scheme, "store" might be expressed in the MTRANS, PTRANS, etc. equivalent of "a place where things are kept such that if you give someone there money, he will give you one of the things". The reason for this is that Schank's understanding programs know about how to deal with "going places", "things", etc., but not with "store". This may be appropriate for normal discourse situations, in which we are rarely interested in the abstract idea of the store itself, but rather in how the store relates to our usual processes of going places, dealing with people, etc. Indeed, this interest in how a fairly standard set of actions and concepts fit into any given context is the foundation of the primitivizing approach: the concepts of interest are posited *initially*, and all deductive mechanism is built around them. Any incoming information must first be resolved in terms of these concepts in order to be understood.

The consultant, however, is not viewing the problem description in this usual sense. He wishes to understand it in terms of his problem-solving model, not in terms of its relation to the world of normal discourse (i.e., not the way "store" relates to sending his child to the store, etc.). I would suggest that the consultant needs the higher-level concepts themselves to be able to see the problem situation in terms of his abstract model. As discussed above, the consultant almost certainly has highly-specialized pieces of "abstraction knowledge" (i.e., the special expert matching tricks mentioned earlier) which might, for example, know the implications of "a store's dependence on rapid delivery" in a business situation. This abstraction knowledge would be *harder* to get at if the higher-level concepts (like "store") to which it is keyed[1] were "disguised in structure" by the primitivizing process.

This basic problem of general semantic techniques arises from the fact that the primitivization process does not know what the problem-solving model is actually *looking for*. It therefore cannot apply all of the specialized tricks and abstraction devices which the expert uses for understanding. In order to do away with this disguising effect of universal primitives, we need some kind of semantic interpretation method which takes into account the needs of the problem-solving model in its effort to understand an utterance.

The second possible use for primitivization suggested above in fact proposes a method that satisfies this need. Instead of having everyday universal primitives, one could imagine a set of *highly-specialized* "primitives" based on the expert's problem-solving model. If one could then come up with a process for directly deriving an incoming concept into these special primitives, the problem description would suddenly be *ipso facto* expressed in the expert's problem-solving terms.

This certainly seems to be a much more attractive possibility than the first one, but I think that it is not feasible. The problem is that any conceivable primitives of the consultant's model will be very different from the usual primitives such as PTRANS. They will be at a *higher* level of aggregation than the incoming concepts, not at the lower level of PTRANS, etc. Thus, instead of being able to resolve a single incoming concept into a

[1]It is keyed this way because it is much more efficient to recognize the manager's concept directly rather than reconstruct it from primitives (just as a chess master keys a defense to a "queen gambit" rather than the set of moves that make it up--he has learned that set of moves as a separate concept to be dealt with, and uses that concept accordingly).

structure of primitives, one would have to conceptualize a structure of incoming concepts as a single primitive--a much harder problem. Furthermore, from my earlier description of characteristics of the manager's problem description, it is clear that even this is a simplification. Incoming structure will not conceptualize into a single primitive, but rather into pieces of several primitives. A whole "primitive" may have to be built out of concepts spread all over the problem description, etc.

For example, suppose I chose "hiring" to be a primitive (referring to the example in I-1). What would map into it?--a one or two sentence description (in managerial terms) involving, perhaps, other "primitives" (say, "backlog" or "training")? Also, perhaps part of the manager's stated hiring policy should be accounted for by, say, a "production costs" primitive in the consultant's model. Thus, instead of being able to resolve the problem description into primitives via a compilation-type process, the program would have to go through some elaborate (and very smart) process of "accounting for" the description in terms of abstract "primitives". All of this seems rather out of place because the suggested "primitives" are not at all primitive with respect to the input language. The approach is backwards.

I don't think reformulation can be implemented by either of these uses of primitivization. In the first case, needed information is virtually thrown away (or made more difficult to get at). In the second, reformulation is traded for a harder-to-implement (because less high-level information is available when it is needed) "decompiling" process. We have strayed far away from the direct primitivization techniques of [Schank] and [Quillian] (which use quite straightforward "compiling" techniques to resolve concepts into primitives. Perhaps systems which use primitivization in conjuntion with other techniques would be more appropriate. Indeed, recently, some researchers have recommended systems *based* on primitivistic techniques for attacking large knowledge-application problems like those germane to reformulation.

These systems are also based on the belief that the knowledge in the system should be organized as a highly-interconnected general structure. For example, Scott Fahlman has recently proposed a general purpose concept net which is queried by a parallel intersection process [Fahlman (May, 1975)]. Also, and especially relevant to my research, there is the "bypassable causal selection network" of Chuck Rieger [Rieger], a method for representing and organizing "commonsense algorithmic world knowledge". I will

just discuss Rieger's work for the moment. The basic idea is to organize an algorithmic knowledge-base into an enormous discrimination network. The algorithms themselves are represented in terms of 26 primitive "links"--a Schankian view of how to express actions in terms of more primitive concepts. Algorithms appropriate for any given situation are found by walking through the net applying discriminating "tests" to the problem environment until the right algorithm is isolated.

Rieger's work attacks a much more general problem than mine, and one might wonder why I don't just use some application of his techniques. In fact, his "theoretical considerations" (see [Rieger], p. 18) are quite similar to mine. However, there are several reasons why his approach is actually inapplicable to reformulation problems. First, to the extent that it relies on primitives, the basic representation suffers from the same problems as the primitivist systems from which it is decended: knowledge becomes "disguised in structure". Now Rieger avoids a lot of this by erecting a discrimination net around the algorithms so that the system usually doesn't have to look at the guts of the algorithm in order to apply it to a problem. Thus, the system relies on there being enough good, appropriately discriminating tests to enable it to choose the right algorithm. Furthermore, the system is capable of introducing shortcuts through the net by combining known test answers. But this use of the discrimination net just makes Rieger's method into a kind of general network approach. It then inherits the reasons why general network structuring is poor for doing reformulation...

The damning initial problem of the distributed knowledge approaches is that it's really impossible to see how to even build their networks for a reformulation-type situation; that is, when expert knowledge is represented by a coherent abstract model. An abstract model is in essence a *point of view* of the problem domain. It is difficult to see, say, for Rieger's scheme, how to express the abstract model as a set of perhaps widely distributed expert shortcuts and tests in a network which must also be used to model the problem at the naive description level. Then, even if the distributed abstract model could be built, it's hard to see how the system could pick through it efficiently to do "special" expert reasoning (which depends on very complete knowledge of the interrelationships within the abstract model). This point applies as well to other distributed-type representations of knowledge, including (perhaps especially) the highly homogeneous intersection environment of a Fahlman-type system.

Another problem is that it would be difficult to make these networks flexible enough to handle the "level of detail" and "model tailoring" considerations I mentioned earlier. The distributed knowledge systems seem to evaluate the consequences of a new finding in too sweeping a manner. In both Rieger's system and Fahlman's system, the effects of a finding by the program are immediately communicated throughout the net via a pre-set general procedure. There is no reason to believe that it would be easy to limit the range (for tailoring) or the scope (for level of detail) of these effects. After all, these network schemes are based on the idea of effecting all of the ramifications of an input throughout the net before taking a look at the piece of structure which is currently of interest.

Finally, it is interesting that both Fahlman and Rieger seem to feel that when their systems become expert enough, they will gain some precompiled "frames" to manipulate rather than having to always wade through individual links. They will then become liable to most of the problems (with respect to reformulation) of the frame systems which will be discussed in section 3.

Many researchers (e.g., see [Minsky], [Brown, A.], [Marr and Nishihara], [Winograd], etc.) believe that programs, instead of representing knowledge in terms of a large, uniformly connected structure, should rely on an organization of the knowledge into relatively loosely connected areas of local concentration. This is in order to maintain graspable, coherent, reason-about-able entities while at the same time reducing the connectivity (and thus interaction problems) between items in the knowledge base. Thus, the knowledge-base is broken into "chunks" of local expertise. Of course, this simply begs the question of how the chunks are to be organized. Consider that it is clear that *something* in the system must be expert about when to use each local expert. The question is, what. Each individual expert? Special expert-calling experts? A smart interpreter? The methods discussed in the next three sections differ mainly in the answers they make to these questions.

Section 2 World-model-base-ism

An approach to building understanding systems which, in complete contrast to primitivism, relies on intimate relations with the problem-solving model, is the "world-model-based" method found in [Woods], [Winograd], [Charniak], etc. These systems are based on a technique that seems to be exactly what I asked for before: input is transformed into a representation which is specifically dependent on the problem-solving model, and is very easily handled by it. In fact, input is "translated" directly into the language of the model--usually a stylized "programming language" for the particular domain. Thus, in this methodology, an input concept like "rapid delivery" from before would be instantly translated into a "program" form which is executed in order to solve some task--finding out how long "rapid delivery" takes, picking up a block, adding to a model of a children's story, etc.

This is usually accomplished by a local expert whose specialty is translating that particular input concept into a particular piece of "program". Most world-model-based systems are firmly in the PLANNER [Hewitt] tradition of having each local expert be cued by the match of some piece of the problem with the expert's special "goal" pattern. The goal is essentially an advertisement of what function the local expert can perform. That is, when a fact is introduced into the database (i.e., when "something happens"), the appropriate local experts come to the fore, do their thing, and depart. Conniver [McDermott and Sussman] and other more recent systems emend this to allow contexts and "possibilities lists"--mechanisms which can be used to provide control over which set of experts may "come to the fore" and which members of the set should take action at any given time. However, in any case, each local expert is expected to know at some level how it interacts with other local experts.

The idea behind PLANNER-based systems has always been that the control over how the local experts are used should reside in the local experts themselves. When one expert is brought in by a pattern match, it can be used to "plan" the activities of other experts--set up calling sequences, make "recommendations", etc. Unfortunately, no existing system has ever been able to really use local experts in this way to provide an effective control mechanism. Instead, most of the "planning" has devolved onto the background failure backup mechanism (see [Sussman and McDermott]), which is certainly

not powerful enough to do problem-solving in a reasonable-sized domain of expertise. In fact, this lack of a real planning mechanism was one of the motivations for the Conniver-style systems. These essentially encourage (force) the system builder to write explicit control mechanisms. The best example of a program built along these lines is the HACKER system of [Sussman], which uses a rather complicated set of expert procedures which arrange plans, mediate conflicts between other experts, etc. This would seem to be a possible implementation for reformulation, with the abstract model being constructed out of a mass of the appropriate local experts. However, there are problems...

First of all, in order to be useful in their full generality (i.e., as metaphors for a variety of real world objects), the concepts of the abstract model must remain spare and manipulable; none of the excess baggage of the real world can be introduced directly into the model. This enables the expert to use the model as a self-contained problem-solving unit that can be applied with maximum flexibility to a variety of problem situations--a key advantage of reformulation over other expert problem-solving methods. In the world-model-based approach, the abstract model concepts are necessarily incorporated right into the local expert procedures which run the actual problem-solving effort. The expertise is buried in the more general problem-solving machinery and is encumbered by the details of the specific problem under consideration as the match-call-and-mediate control structure rolls along. It is very hard to get at this distributed and procedurally represented expertise in order to do the kind of explanation and abstract model reasoning required of a *reformulation expert*. I think that the expert would much rather maintain the integrity of the abstract model and develop a separate battery of techniques for abstracting descriptions into the model--the experience-based "abstraction techniques" mentioned earlier.

Next, world-model-base-ism would sacrifice the actual implemental decoupling of the problem domain and the expert domain which is one of the nicest features of writing programs in the reformulation paradigm. Because they connect input directly to local experts, and because experts are tied to other experts via rather complex procedural interaction, everything in a world-model-based system tends to become inextricably interconnected. Even without the inevitable implementation problems it causes, I think that this combining of the input and expert conceptual schemes actually *complicates* the problem-solving task. Weaving the two processes together tends to create a tangled mess of conceptual (and thus, automatically, *procedural*, for the world-model-bassi) interdependencies that is almost impossible to sort out for the program or the programmer.

Finally, there is the matter of being able to handle global considerations like model tailoring and choosing level of detail. In the world-model-based scheme, these would have to be handled by interactions between local experts--controlled locally by the individual experts. It's hard to imagine how these experts could stay usefully local and still be capable of handling these sophisticated model-wide considerations. Unfortunately, the advances of the HACKER approach do not really seem to aid here. While it does provide some real medium-range planning capability, it does so at the cost of binding the modelling and problem-solving knowledge which makes up that planning expertise into tight little packets of procedures which are almost impossible to use more globally to tailor the model, control level of detail, etc. In other words, they make it even harder to see the big picture (see the comments about this in [Sussman]). For a reformulation expert, this "seeing the big picture" means using the abstract model in its fullest generality as a separate reasoning environment--an essential part of the reformulation approach to problem-solving.

Therefore, I think that, in terms of implementing reformulation, world-model-based systems are unsuitable because they necessarily introduce an unfortunate restructuring of the problem-solving knowledge. This change--the homogenization of the abstract model knowledge into the problem-solving mechanism and the input domain data--not only makes it difficult to apply the expert knowledge in the places it should be applied, but also complicates the implementation by introducing tight interdependencies where there could be more slack, and by forcing fundamentally global information to be expressed locally.

Section 3 Recognitionism

The currently popular "frame recognition" approach (see [Fahlman (Dec., 1973)]) eases up some of the constraints of the world-model-bassi. We will see that the approach separates out a "fitting" process from the overall problem-solving process. The fitting process is used to connect the input and problem-solving domains, and is thus certainly a possible implementation for reformulation. However, I think that the "connection" dictated by the recognition method is the wrong kind for reformulation.

The recognitionists' approach depends on the fact that many understanding systems, such as vision, medical diagnosis, etc. are developed by forming generalizations of the objects of the real world of the domain[1]. The expert (or, indeed, any practitioner) structures his model of the domain around these generalizations by forming *frames* (see [Minsky]) which act on (expect, debug, explain, etc.) members of a particular class. (A "class" is just the category which is the generalization of a set of real objects (e.g., "chairs", "liver ailments", etc.).) An object is recognized as a member of a class by more or less straightforward matching of the input to an appropriately represented exemplar of the class. This matching process is very different from reformulation--a doctor rarely "stops and figures something out": he either knows that he knows something about it or that he doesn't after a few relatively shallow transformations (see the protocols analyzed by [Miller] for more on these transformations, and [Rubin] for a theory of the recognition process that combines them). His knowledge base (*cum* knowledge-application methods) is broad and shallow; this is the essence of recognitionism.

The frame theorists are essentially the inheritors of the world-model-based approach to organizing local experts. However, they introduce some new wrinkles. Although they are rather ambivalent on this point, they clearly believe something along the lines of frames being responsible for pulling in other frames (perhaps with some sort of context mechanism included). Frame-recognition systems work by filling slots in frames, calling in new frames when indicated by some condition on the slot-filling mechanism. As McDermott has pointed out [McDermott], frame systems essentially use this slot-filling mechanism as a smart calling method for function application. That is, the system attempts to fill slots to see whether the functions of a particular frame can be appropriately applied to the problem. If an exceptional condition occurs during slot-filling, another frame is called, etc. (see [Kuipers], [Rubin]). Most of the effort seems to go into deciding which frame should be called when--and these decisions are effects of the slot-filling mechanism. Therefore local chunks interact with each other via an overall control structure which is a *result* of this "calling" and message-passing between frames (again, see [McDermott]).

This sort of structuring is not really applicable to reformulation. The reformulation expert's abstract model of the domain was not developed by generalization,

[1]However, Marr's system [Marr and Nishihara] deals with the domain of vision processing via what I consider to be reformulation techniques, as I will discuss more fully later.

but rather according to a self-consistent methodology perhaps even taken from another domain (e.g., control theory, statistics, etc. for business consultants). For example, consider the following scenario...

Suppose we walk into a room and are asked to find the telephone. We start by looking for something which is rather small, black, shiny, is sitting on a desk, has a cord running from it, etc. We are essentially using our "telephone frame" to help us recognize the object we are searching for. Now this frame may be quite complex: we are not fooled by wall phones, "Princess" phones, blue phones, etc. The recognition process itself, the "matching to the frame", is , however, relatively straightforward and efficient.

Now suppose we cannot find the phone; nonetheless, we are assured that it is there, but that it is in an unusual form. Our usual recognition process will no longer serve us in good stead[1]. But, if we are expert enough, that is, if we have a high-level model of "telephone", we can switch to an expert reformulation process to find what we are looking for. Thus, keeping in mind the functional structure of a telephone, and using our knowledge of electronics, common sense, etc., we might say "What can be used as a receiver? Must it use mechanical vibration over electromagnetic contacts, or is another mechanism possible? How big or small can it be? What material can it use? Should we spend time looking for a cord, or can the signal be broadcast?, etc." (The functional reasoning used here is of course not the only kind of "expert reasoning" possible, as we shall see.) In this way, if we know enough, we can find a "telephone" even if it is in a very unusual form (e.g., the whole room might be a telephone). Note that this is very different from the earlier recognition process: we are not matching the exemplar of a category, but instead are forcing a new kind of real object into a well-known *independent* (i.e., based on electronics, etc.) abstract model. The model is not built around recognition frames, but around its own logic. Thus, there are no generalized categories or "frames" for objects in the problem description to *fit* into. Instead, the expert must *reformulate* the real world objects in terms of his model by using sophisticated abstraction techniques. This abstraction is much more like a problem-solving process than a straightforward recognition process.

[1]Of course, most people in this situation would try to use it anyway, since they have nothing else. They might thus look for a disguised or distorted normal phone, not realizing which features of a telephone are necessary to its operations and which are simply conventions (e.g., they might search for a disguised handset, not realizing that the parts may be separated or in fact changed into very different entities).

The more complex "find the phone" problem is an example of reformulation. I suggest that it is very different from the recognition process exemplified by the first part of the "find the phone" problem. The reformulation expert reasons in terms of his model. The expert knows a great deal about how a piece of the model works; he just has to find out how that piece of the model is represented in each particular problem description. Furthermore, he uses the model itself as a reasoning tool to aid him in this process of association of model pieces to input pieces and to do any necessary problem-solving in the domain. The recognitionist does not expect the collection of frames into which a particular situation fits to form a self-consistent reasoning environment. Indeed, very little reasoning is done in terms of more than one frame: each frame is designed to simply solve a particular part of the problem in terms of whatever real-world object instantiated the frame. Most of the expertise has gone into making this instantiation--once it's done, the problem-solver in the frame is usually "ready to go" without further ado. This is why I earlier called the use of frames a smart calling method for function application.

It is certainly true that many experts use just recognition (in a diagnosis-type process) or recognition in combination with reformulation. Thus, the processes can certainly coexist, and recognition can be *part* of a reformulation scheme (or vice versa). Nonetheless, as in the telephone example, the processes are clearly different. As was the case with world-model-base-ism, an attempt to implement reformulation *as* a recognition process would force serious changes in the problem structure. One would have to superimpose a structure of frames onto the abstract model such that pieces of the problem description could "fit into" pieces of the model via a matching-type mechanism (note that the "abstraction knowledge" has disappeared). The frame structure is especially hard to build in this case, since it involves homogenizing two very different pre-existing ways of looking at the world (rather than *building* a generalization of one way of looking at things)[1].

Thus, recognitionism, though natural to many kinds of problem-solving, is wrong for reformulation. Building the "right" frame structure is just too difficult (if not

[1]Nonetheless, some consultants actually do this to a small part of their reformulation process: a great deal of experience with solving certain kinds of problems allows them to simply "match" certain problems directly to solutions in cases where they formerly used reformulation. The result of this "recognitionizing" is that the consultant can solve problems within this narrow range much more efficiently, though more rigidly. Such consultants often then specialize in just those problems within their "recognition range".

impossible) in the face of the constraints posed by the differences, discussed in I-3, between the conceptual domains of the problem description and the abstract model. Furthermore, a frame-structured approach to reformulation seems to ignore features of real world consultants who use reformulation (e.g., their "abstraction knowledge" and their self-contained, independently explicable model). Finally, there is the fundamental difference in the problem-solving approaches of recognition and reformulation, as exemplified by the "find the phone" problem.

I'll keep looking.

Section 4 Straight-shooting

The final major approach to "doing understanding" has important similarities to those of the world-model-bassi and the recognitionists. In fact, it could be said in a crude sense to be a "cross between the two". Nonetheless, it has a separate philosophy and methodology which make it very relevant to the kinds of problems I need to solve.

In terms of the local expert question, straight-shooters are the interpreter folk, who believe that instead of allowing the control structure to just sort of rise out of what the frames are doing (or something), local experts should be accessed for specific problems by an interpreter which knows how to put them together (perhaps in response to more global explicit "plans").

The basic straight-shooter philosophy is that the problem domain and the problem-solving model should be carefully digested as a first step toward implementation of the understanding program. This "digestion" process consists of thoroughly compartmentalizing the relevant knowledge of the field, with heavy emphasis on categorization of the world objects by generalized "type" (in this way, similar to the recognitionists). The overriding theory of the analysis is "a place for everything, and everything in its place". Thus, representational issues become extremely important (there must be a well-defined *place* for everything). After this digestion phase, the whole domain is hopefully broken down into short chains of actions connecting small groups of things. That is, if the domain has been successfully digested, there will be only one or two possible "things to do" at any given time during the problem-solving effort.

Assuming this properly digested domain, a very straightforward program is written--essentially a large interpreter which "calls" the individual routines that know how to handle a given piece of knowledge. Thus, the action of the program is first, put any input into its proper pre-determined compartment (via generalization, a kind of pattern-matching, and, especially, strong built-in notions of what to expect next (see [Martin])); and second, call the piece of program that knows what to do with information in that compartment.

The method essentially assumes that the whole domain of the program has been modelled once and for all. That is, a generalized (rigid) framework of what can be in the domain has been built into the program. An incoming problem description must therefore be an *instantiation* of some of the built-in concepts. This is why the program always knows pretty much what to do about things.

Thus, the straight-shooters present a method which is a step away *from* the world-model-bassi in that input is not directly compiled into semantic interpretation routines. On the other hand, they do not move a step closer to the recognitionists, first, because they see fit to digest the domain rather than taking it as it is, and second, because they specifically avoid the distributed control structure and general prevailing mysticism of frames. In some ways the fitting process of the straight-shooters is more constrained than that of the recognitionists. However, comparison is difficult because the concept of "fitting" itself has changed. Since both the problem-solving model and the domain have been pounded into a shape that allows instantiation, "fitting" (by instantiation) becomes the *given*; the model and the domain are then represented to suit.

The straight-shooters' point is that whether or not you believe people do it, working out the grand representational scheme is the best way to write expert computer programs; i.e., that experience has shown that that methodology is the most amenable to implementation. (Some straight-shooters will argue that people use this methodology in their own thinking processes; others won't.)

Specifically, this approach has been the basis of most of the "conventional" computer programs (i.e., almost everything that works) written since the world was young. In addition, the straight-shooters can point to "unconventional" (usually "natural") problems (like finding the 3-D picture from a set of lines [Waltz] or mathematical integration

[Moses]) which were worked on for years with fancy "unconventional" methods and then eventually solved by straight-shooting. Recently, the straight-shooting approach has been brought to bear on the problems of consulting ([Krumland], [Bosyj], [Shortliffe]) and natural language processing ([Martin]).

Although these systems represent very different problem domains and implementation styles, each illustrates the basic advantages and limitations of the straight-shooter approach. Krumland is interested in applying (via an instantiation process) small generalization-type linear models to solve problems presented in dialogue with the manager. His system is an extension (albeit a major one in terms of the program's "world knowledge") of the more prosaic "on-line management information system". Characteristically, his research problems center around representational issues for the models and the dialogue. Bosyj has constructed a computer implementation of an existing operations research consulting questionnaire. The program contains a rigid, basically hierachical model of the firm. It works by allowing the user to specify "values" for its "facts"--i.e., to instantiate the general model of the firm. The user is essentially particularizing the model to his firm. Shortliffe's MYCIN system is based on a production-rule-*cum*-numerical-scoring methodology and works in the area of diagnosing and treating bacterial infections. Once again, the user is asked to characterize his particular circumstance in terms of the program's model of the domain via a rigid set of questions.

Since the MYCIN system is probably the most familiar of this batch, I will use it to illustrate the consequences of using the straight-shooter approach to building an expert system. The first point to note about the MYCIN system is that it does a fairly good job for what it does. This is very characteristic of the straight-shooter approach. In fact, so much so that some observers have suggested that straight-shooter techniques, especially MYCIN techniques, can be the basis of "any" expert system. I think that this enthusiasm is misplaced. MYCIN and the other straight-shooter systems work because their areas of application are well chosen to be digestable, and because a lot of work has been put into the digestion phase. Note that I kept saying in the previous paragraph that the user characterizes his problem in terms of the system's model. By this I mean that the expert's knowledge and the types of user problems have been so carefully structured and compartmentalized (i.e., digested) that the system can ask relevant questions about its model which are perfectly comprehensible to the user, that the user can answer these questions in terms of his particular problem, that the user can characterize most problems

of interest in the domain with his answers, and that the end product of the characterization session is something that the expert system knows very well how to deal with.

Certainly this seems to be all that one can ask of an expert system. Yet there is a major drawback to this approach. That is that in every existing straight-shooter program, significant sacrifices in the form and content of the expert model had to be made in order to make the digestion and implementation work. MYCIN is an excellent example of this. Although real life symptoms are not in general independent in the probabilistic sense, they must be so in MYCIN. Thus, the expert must either convert his symptoms into a collection of "super-symptoms" (i.e., knowledge about a particular symptom is broken down into several separate chunks differentiated by the addition of extra conditions on their applicability--sometimes resulting in incredibly particularized pieces of the knowledge base) or be willing to live with them being treated as independent. Although it is known that the reasoning process used by the experts differs in important ways from the numerical scoring process (see [Miller]), experts must state numerical "certainty factors" for each of their inferences, and rely on the built-in combination schemes of MYCIN to reason (numerically) in terms of those inferences. That is, the experts provide MYCIN with individual rules of inference (i.e., "when you see piece of evidence x, suspect bacteria y with 85 per cent confidence", etc.), but must rely on MYCIN to combine these rules to work on any particular task. The problem is that there is no reason to believe that MYCIN's way of combining these inferences is anything like the expert's way, and I personally doubt that very many experts understand exactly what MYCIN does with the knowledge they provide to it. MYCIN forces other kinds of reshaping of the expert knowledge, but these are the major ones.

Note that the effect of these restrictions goes beyond minor irritation to the experts or an admission that there are "a few cases in which MYCIN does the wrong thing". First, there are serious questions about the extensibility of a straight-shooter system like MYCIN. What happens when the expert has to push on the limitations a little more, or really has to go beyond one limitation entirely? Here is where straight-shooter systems really pay the price. Since the whole system is built around the carefully digested model, raising one key restriction could, and probably would, cause the *whole* methodology to collapse. This is why straight-shooters must be so very careful to do everything right in their initial digestion phase. Second, there is a more subtle point. How can one be sure that the straight-shooter system is really working? After all, it no longer works in terms of

the expert's representation and reasoning mechanism. It is doing something else. Now that "something else" certainly works most of the time, but what hidden assumptions is it using? How does its decision process differ from that of the expert? These are questions that are very difficult to answer in many straight-shooter systems. The inclusion of explanation facilities doesn't help. In fact, it can hurt. The program can only explain its internal methods--the hidden differences from the real expert model are not brought out. The system, after all, knows nothing about the expert's real model, has no idea how the expert thinks. If the expert or user places too much confidence in these explanations, he may be misled into thinking that the program *really* knows what it is doing in the genuine expert sense. (Everyone knows enough not to trust a program which can't give explanations, but they may not know enough not to trust one that can.) For example, MYCIN can explain all about the certainty factors that were attached to its chunks in the first place, and its current level of certainty in any given hypothesis. However, it cannot explain the hidden assumptions about logical subsumption, independence of symptoms, "loop breaking", etc. which went into determining the level of certainty for that hypothesis given the initial certainty factors. Finally, I think that no matter how well the domain is digested, straight-shooter systems will tend to be rigid with respect to user input. The user is never asked to simply "talk about his problem", but rather to characterize it in terms of the digested model. While this model is presumably pretty close to what the user wants to talk about, it's not the same as giving the user full freedom of expression and making the expert figure out what to do with what he's told--the operating mode of many consultants.

My overall conclusion about all of this is that the universe of techniques for expertise application represents a gradation of susceptibility to straight-shooter techniques. On the one hand are problem-solving methodologies in which straight-shooting is perfectly okay--no significant sacrifices have to be made in order to make things work. Then there is a continuum of methods in which more and more sacrifices have to be made in order to implement the expert model using current programming techniques. Finally, there is the range in which it hurts too much to make the sacrifices--straight-shooting won't work.

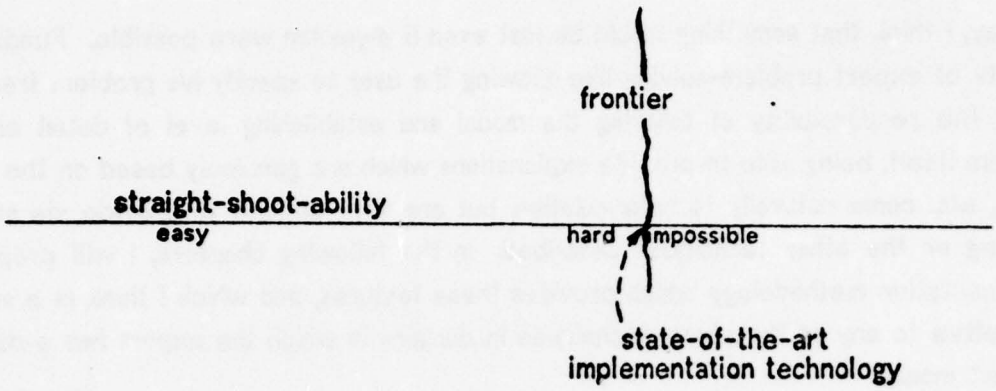


Figure 8. Continuum of straight-shoot-ability of expert problem-solving techniques

It is interesting to note that the "frontier" of straight-shooting moves outward as "conventional" implementation methodologies improve. The OWL system of [Martin] represents an attempt to produce a significant advance in the implementation technology for expert systems based on a carefully worked out representation scheme for natural language data. The idea is to push the frontier of straight-shooting out far enough that the implementers of any expert system can use the straight-shooter-based OWL representation as the basis of their expert model, and then add the more specific features of that particular domain of expertise in terms of generalized OWL implementation tools. It is still far too early to tell whether something like the OWL system can be built or whether expert system implementers will really be able to use its features as a core for their programs without making serious sacrifices. Nonetheless, this effort represents a sort of "meta-straight-shooter" approach to the problem of building expert systems.

At any rate, as you can undoubtedly tell by what I've said so far, I think that the reformulation methodology is beyond the pale of straight-shooting techniques. The whole modus operandi of a reformulation expert is based on the fact that his model is a *separate*, self-consistent reasoning environment. To digest it would be to destroy it. Furthermore, it's really hard to see how to digest something like Newtonian physics or the "workforce need" theory mentioned in Chapter I into something that the user could instantiate.

Anyway, I think that something would be lost even if digestion were possible. Fundamental aspects of expert problem-solving like allowing the user to specify his problem freely and taking the responsibility of tailoring the model and establishing level of detail onto the program itself, being able to provide explanations which are genuinely based on the expert model, etc. come naturally to reformulation, but are very difficult to provide via straight-shooting or the other techniques described. In the following chapters, I will propose an implementation methodology which provides these features, and which I think is a superior alternative to any of the above techniques in domains in which the expert has a definable abstract model.

Chapter III
Of Alfred

The implementation I use in my program is different from all of the ones considered so far. While the knowledge is organized in terms of local chunks, the chunks never call each other directly, nor are they designed for access and control by a central interpreter. Instead, the explicit knowledge they contain is selectively utilized according to the needs of the abstract model, to be used in a way defined by the dynamic needs of the reformulation process. Specifically, the chunk *setup* mechanism, which is the major problem-solving element of the program, is controlled by the process of associating a particular chunk with a piece of problem description and using that association to build up the actual model of the problem. This is done by making all of the key problem-specific modelling knowledge learned in doing the associating into a special "problem-solving context" which is then fed back to the chunk setup mechanism to enable it to tailor the abstract chunks of the model to the particular problem at hand. This feedback process and the structure of the problem-solving context are the essential contributions of the reformulation theory to the technology for implementing expertise in programs.

The **abstract model** which is the fundamental knowledge-base of the program consists of a structure of chunks like the ones we saw in Chapter 1. It does not differ from other chunk-style knowledge-bases except in one important particular: its structure is indigenous to the expert's model of the domain and has a semantics of its own; the model has not been divided for the convenience of the control structure. This underlying semantic structure of the chunks is important for making the whole modelling mechanism work, as it provides the basic outline for deciding which chunk to use when. Chunks which are directly connected to the sections of the model which have already been associated are usually the ones that ought to be associated next.

We can get some insight into the way that chunks are organized within the model by looking at the work of Kuhn. In his famous *The Structure of Scientific Revolutions*, Kuhn examines the problem of how scientists try to see the phenomena of nature in terms

of their pre-set abstract models[1]. He decides that a scientist views the world through a "paradigm" which groups and explains the phenomena of nature in a certain way. The *scientist uses his paradigm to set up working problems for the articulation of that paradigm* (that is, the inclusion of more natural phenomena into the explanations of the paradigm). The paradigm then provides certain rules and methodologies for solving these working problems or "puzzles". The working problems are puzzles because of the "assured existence of a solution" (the scientist only sets up working problems he *knows* (within the context of the paradigm) he can solve) and because the techniques for finding the solution are governed by clearly defined rules (this is especially important, since, as in any puzzle, the solution may be trivial if the rules are broken). The work of the scientist, which is clearly reformulation, is thus closely akin to *puzzle-solving.*

Now I feel that this provides an excellent picture of how the consultant discussed in section I-1 reformulates problem situations into his model. The paradigm of course corresponds to the consultant's model. The consultant will always view problem situations in his area of expertise through this model. Furthermore, he uses the known structure of the model to go about the task of formulating the problem situation in terms of the model: he sets up "puzzles" of the form *abstract and simplify some part of the problem so that it corresponds to this piece of my model.* Just as for the scientist, the way in which pieces of the problem situation may be abstracted into the conceptual scheme of the model is governed by a set of rules which are part of the model. The experimentation process of the scientist corresponds to the dialogue-driving process of the consultant. Thus, just as the scientist designs experiments to provide information necessary to solve a certain puzzle, *the consultant designs pieces of dialogue to elicit information from the manager.*

Note, then, that a "finished" paradigm can be seen as being made up of a set of solved "puzzles" which maintains the structure that originally formed it, i.e., a record of the way that each puzzle caused the next one to be set. I have copied this notion exactly in implementing the abstract model. Its chunks correspond to puzzles, structured such that the effort of associating one chunk to a piece of input leads to the setting up of the next chunk.

[1]This is of course the activity of "normal science" as defined by Kuhn, not the revolutionary science to which most of his study is devoted.

Because the model is well-structured, and because the individual chunks are at a very abstract level, most of the work of association is involved, not in choosing the right chunk to use at a given time, but in changing the "right" chunk to suit the particulars of the problem at hand--in *making it into* the right chunk, if you will. Note that this is very different from frames, which make associations only to see if a diagnosis fits, if an object is recognized, etc. That is, frames make associations to see if a certain program function is applicable--and incidentally to supply it with arguments. For reformulation, making an association is part of the process of building a separate environment in which to do expert reasoning for that problem. At any given stage, then, the association process is taking something from an unstructured world (the problem description) and placing it into a structured world (the chunks that have been associated so far--the modelled system) in accordance with a structured set of rules and procedures for doing this (the abstract model). The association effort for one chunk interacts with the association efforts for all of the other chunks via the cumulative process of building the modelled system[1].

This interaction is by no means an unstructured by-product of the model-building process. Instead, it is specially (in fact, *expertly*) controlled by that process via the "problem-solving context".

When an associated chunk of the model is put into the modelled system, the model of the problem so far, a special part of the program ferrets out and appropriately records the globally interesting aspects of the new chunk. That is, it not only builds up the modelled system that the program will use for problem-solving, but also maintains the special contextual information within the modelled system which is used to guide the selection and setup of new chunks. One might ask what this hopes to accomplish, since it is just another "something" that knows about local chunks. The advantage here is that the mechanism that incorporates the chunks into the modelled system has at hand the current

[1]It seems to me that many of these same considerations are behind the design of the vision system of [Marr and Nishihara]. In fact, I believe that this program is another example of an expert system which uses the reformulation methodology, complete with abstract model (of 3-D objects) and modelled system. Although the reformulation philosophy is not explicitly stated in it, and although the implementation does not use the same feedback through the modelled system that I will stress as key in later chapters, the use of the abstract model itself and the overall "connection" philosophy (see Chapter VI) are quite similar to those described here. I think that a comparison of these two reformulation-based expert systems in such different domains is very enlightening.

state of knowledge about the problem so far, as structured by the abstract model, as expressed in a specialized contextual form. That is, since the expert is modelling the problem in terms of the abstract model, the result must in some sense be a reflection of that abstract model. That "sense" is in fact that the *interaction knowledge*--information about how one chunk affects another--contained in the abstract model has been retained in the model of the specific problem. This is the real "expertise" of the abstract model: the details of local chunks may change considerably, but the interaction between them is still essentially the same. The reformulation approach has the advantage that all of this expertise which has been built into the abstract model (by centuries of effort, the insight of a single consultant, or whatever) is retained intact. In other words, when it comes time to handle the interaction of local chunks, the program knows everything it can know, and it knows a lot about what it knows because what it knows is in terms of the abstract model.

The purpose of the special contextual form maintained by the incorporation mechanism is to provide an easily readable format for condensing the appropriate information from the modelled system. This saves the program the trouble of actually having to look through the detail of the modelled system every time it needs to know something. Recording contextual information in a special format also allows it to be organized in a way that is particularly useful for guiding chunk setup activities. Information for controlling the overall direction or mode of problem-solving activity--that is, the top level goals of the system for each problem-solving effort--are maintained in the **theme**. As we will see in Chapter VII, themes exert a very important influence in defining and delimiting the modelling task, since the theme of a problem-solving effort is always based on the presenting symptom of the problem description for that effort. Information at the level usually associated with contexts is maintained in the **trend**, though, as we will see in section 2 of this chapter, this information often differs radically both in form and function from that used in usual context mechanisms (e.g., Conniver--see [McDermott and Sussman]). Finally, information from a previous association which affects only the next chunk to be associated--the piece-to-piece connectivity information for the model--is maintained in the **edge**.

The contextual information stored in the theme, trend, edge mechanism is the basis of the associate/incorporate feedback link shown in figure 7. That is, it provides the medium through which global interaction information influences problem-solving activities. The exact locus of the feedback link is in the mechanism which sets up the next chunk for

consideration. Theme, trend, and edge information is used to "edit" the chunk so that it reflects what is known about the problem. It also restricts the use of association procedures, monitors the level of detail of the modelling effort, etc. The point is that while individual chunks never have to know anything about this global inheritance, it actually determines which chunks will be used, the exact form in which they will be used, and the particular way in which they will be used.

This associate/incorporate methodology provides the fundamental design principles for the consulting program (known as ALFRED[1]) which will be described more fully in the rest of this chapter. The basic action of the program is as follows (see system diagram on the next page):

-1- The chunk setup mechanism chooses a chunk to be processed, edits it according to theme, trend, and edge information, and places it "under consideration".

-2- Abstracters look for a piece of the problem description which they can associate with the chunk; if no match is found, the chunk is restructured and the matching effort is tried again.

-3- If an association is found, any characteristics of the association which might impact the rest of the modelling effort are abstracted for the theme, trend, and edge as the association from step -2- is incorporated into the modelled system.

[1]"Alfred" is Old English for "good counsel" (derived from "elf counsel"). If you feel you must have an acronym, it can stand for "A Laboratory For Reformulating English Descriptions". If you care to go further, "-red" is another form of "rad", which comes from the Old English "hraed", a word whose meaning is preserved in the modern, though rather rare, "rathe", meaning "early in the season or period". This suggests the prototype nature of the program. Finally, "alf-" derives from Old English "aelf", akin to Middle Low German "alf", meaning "incubus". You can work that in somewhere yourself, if you'd like. I will feel free to refer to the program as "Fred", "Freddy", etc., as the occasion demands. To keep things fair, the program should also be thought of as "Alfreda" ("Frieda", "Freda", etc.) about half the time. Cheers.

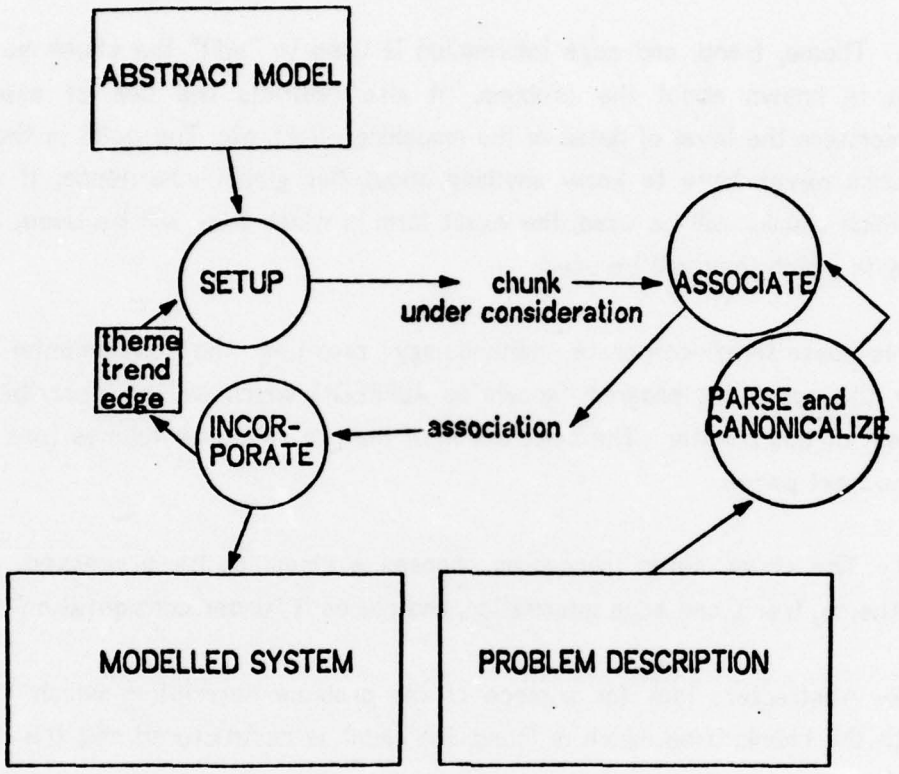


Figure 9. Overview of program action

This flowchart gives the major program functions and data structures. More detailed flowcharts of the chunk setup and association processes appear in the section below. In addition, the table shown on the following three pages lists all of the constructs that will be discussed in this thesis along with their implementations and possible properties. This information should make it easier to gauge the scope and complexity of the program activities that are introduced in this chapter and discussed in more detail in the rest of the thesis.

Now, taking it step-by-step, let's firm up the ideas presented so far in this chapter in the context of the Alfred program.

entity represented by a LISP atom
possible properties:
(1) a list of states that the entity can be in
(2) a list of actions that can act on it
(3) a list of abstracters that can be used to find matches for the entity
(4) a list of elaborations
(5) a list of characteristics of the entity
(6) a flag which indicates that the source of the entity need not be modelled (e.g., not necessary to find the source of (PEOPLE))

action represented by a LISP atom
possible properties:
(1) a list of entities that the action can act on
(2) the state it puts the entities into
(3) a list of abstracters that can be used to find matches for the action
(4) a list of elaborations (often referred to as "policy choices")
(5) a list of characteristics of the action

In addition to these built-in properties, any action or entity may be given the following properties in a particular model:

property	*value*
MODIFICATION	a list of entities (characteristics are often modified by (POSSIBLE) or (NECESSARY))
PER	(DAY), (MONTH), (SUMMER), etc.
AGENT	(FIRM), (CUSTOMER), etc.
OF	(FIRM), (CUSTOMER), etc.

Table 1. Basic Elements of Alfred's Representation Scheme

modelling entity represented by a LISP atom
possible properties:
(1) a default "how to go about modelling me" procedure
(2) inference rules for causal reasoning about the modelling entity, used primarily by the SHOW-CAUSE PURPOSE

The following terms are used to describe a piece of model structure as it is being processed:

"chunk" a list of elements such that:
(1) the first element is a modelling entity
(2) the rest of the elements are either:
(a) lists of the same format
(b) actions or entities
the second element of the chunk is termed the "object"

"piece" any element of a chunk except a modelling entity by itself; note that a piece may be, but is not necessarily a chunk

(E.g., given the chunk

```
(DETERMINED-BY (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
      (INFLUX (ACT-ON (PEOPLE)
                      (HIRE)))
      (OUTFLUX (ACT-ON (PEOPLE (EMPLOYED))
                       (FIRE)))),
```

Table 1. Basic Elements of Alfred's Representation Scheme (cont.)

the object is (NUMBER-OF (PEOPLE (EMPLOYED)) NIL), and some pieces of the chunk are (INFLUX (ACT-ON (PEOPLE) (HIRE))), (ACT-ON (PEOPLE) (HIRE)), (PEOPLE), etc., but not DETERMINED-BY, INFLUX, etc.)

Any chunk or piece may also be given the properties MODIFICATION, PER, OF, and AGENT. In these terms, other constructs in Alfred have the following structure:

construct	*value*
Theme	a list of one or two chunks
Trend	
TOWARD	a piece
PURPOSE	a piece
LEVEL	a list of actions and entities
ABBREVIATION	a list of pieces
Edge	an entity

Table 1. Basic Elements of Alfred's Representation Scheme (cont.)

Section 1 The abstract model

The basis of the consulting program is the abstract model, the representation of the self-consistent high-level model of the consultant's domain of expertise. In keeping with the philosophy set down earlier, this representation is as "natural" as possible. That is, I have not compromised the integrity of the abstract model for the sake of other parts of the system. In order to do this, I have closely followed the (relatively few) clear explanations by consultants of the models they think they use. Also, I have used textbooks to get at the models consultants and scientists want to teach their students. ([Kuhn] gives a nice set of reasons for why the paradigms of science are found in textbooks.) I have then tried to apply my findings to the area of cause-effect modelling of business systems, concentrating on the dynamic interactions of parts of the firm. I have come up with the following structure for an abstract model in this field.

The model is organized by **theories** (like the "workforce need" theory of section 1-1), which represent a specific point of view of how certain parts of the firm interact with each other. Attached to each theory is a **problem-definition** which describes the firm situations to which the theory is applicable. (The problem definition for the workforce need theory says that it can deal with workforce fluctuations caused by actions of the firm.) Ideas for policies to correct system dysfunctions are also attached to the theory. That is, for example, attached to the workforce need theory are a variety of valid hiring policies. If the hiring policy of the firm is found to be dysfunctional, a different policy is suggested which won't cause the dysfunctional behavior. Thus, for any kind of problem which the model can attack, the theory defines the parts of the firm that need to be considered, the way they interact, and the way they are affected by policies. Note that a theory represents only one point of view; several theories could be applicable to the same firm situation (and deal with it in totally different ways).

The largest conceptual units within a theory are **sectors**. They refer to coherent parts of the firm or its environment as seen by the consultant. One sector, for example, is the workforce (or "labor") sector that is a key aspect of the workforce need theory. It consists of the representation of workforce flow shown in figure 5, along with the representation of the hiring and layoff policies, models for the delays, and all of the knowledge attached to these representations. Other sectors of importance to the

workforce need theory are production and customer ordering. Furthermore, as we will see in a second, sectors represent divisions of the macrostructure which become boundaries in the microstructure. That is, the consultant models the system by modelling sectors and putting them together. The pieces within each sector still maintain a special relationship with each other after they have been modelled, forming what I will call a "close interaction group".

One of the principle problems for the consulting program is that any given sector requires widely different representations depending on the particular firm and the purpose of the model. The representation of the sector must somehow make explicit what the model has to say about that particular part of the firm: it must represent the "essence" which the consultant seeks to capture (returning to the terminology of section I-1). The embodiment of this "essence", usually the major flow sector in these cause-effect flow models, is called the **basic idea**. It is the salient piece of structure *around which* the rest of the sector is constructed. This basic idea is the key to modelling the sector. The sector is a collection of actions and entities built around the single basic idea construct. The basic idea does *not* correspond to an exemplar or scenario for a frame. It is not a "typical member" or "identifying concept" or "usual action", but rather a key piece which must be identified in terms of the problem description for the rest of the reformulation effort for that sector to proceed.

For example, the basic idea of the labor sector of the workforce-need theory is that "people are brought into the firm by hiring and taken out of the firm by layoff, subject to delays", or as shown in the first chapter,

```
(DETERMINED-BY (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
    (INFLUX (DELAY1 (ACT-ON (PEOPLE)
                            ((FIRM) (HIRE)))))
    (OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                             ((FIRM) (LAYOFF))))))).
```

This simple notion of flow of personnel through the firm is the key to the rest of the reformulation effort in this sector. If Alfred cannot see how it applies to the organization at hand, that is, cannot associate it with the problem description, it cannot "get a handle" on this sector; if it can, it's well on its way.

In the example of Chapter I, I introduced the notion of two different kinds of objects in the abstract model. One kind consists of **entities** like (PEOPLE), (ORDER), (PRODUCT), etc., and **actions** like (HIRE), (PURCHASE), (TRAIN), etc. These are the expert's concepts of what is in the firm. That is, they are the things that he considers to be coherently describable and useful to his model. Attached to each of these entities is all that the expert model knows about that entity. For example, all that Alfred knows about (PEOPLE), which isn't much, is that (PEOPLE) is an entity which can be associated with people, men, or prospective employees, that people can be acted on by the (HIRE) activity, and that it is not necessary to figure out where (PEOPLE) come from (as it would be necessary to figure out where (PRODUCT) comes from) when the problem is modelled. Alfred's knowledge about (HIRE) consists of the fact that it is an action which operates on the (PEOPLE) entity and then goes on to give several different policies for controlling that action. As mentioned in Chapter I, these are in the form of elaborations of the various pieces in the hiring policy chunk. Entities, and especially actions, usually have several alternate statements which are appropriate in different circumstances. Again, as I said earlier, there is no knowledge of when the various alternatives should be used, only the collection of possibilites.

The other kind of objects in the abstract model express the structural relationships between entities and actions. These are called **modelling entities**. The DETERMINED-BY construct we saw in Chapter I is such a modelling entity. As with actions and entities, all that the expert model has to say about individual modelling entities is attached directly to those modelling entites--it is never found elsewhere. The kind of knowledge attached to a modelling entity tells how it should be used, as we saw in the case of DETERMINED-BY , and in some cases, how to reason in terms of it. An example of a "reason-about-able" modelling entity is FLOW, the explicit expression of the kind of flow relationship that was implicit in the DETERMINED-BY structure of Chapter I. Attached to this modelling entity is the information we saw used in the Chapter I example: "anything that flows in must flow out", "actions in the flow affect things downstream", and the like--knowledge about how to figure out cause-effect relationships within the flow. This kind of reasoning is basic to the problem-solving approach of the program, as we will see in Chapter VIII. The complete rules for reasoning about FLOW are given in Chapter V.

A more concrete inventory of abstract model knowledge and the way it is represented is given in Chapter V. For now, the only important thing is that knowledge

about pieces of the model, whether it is association knowledge which tells how the piece connects to the problem description, or modelling knowledge which tells how the piece should be used for problem-solving, is attached directly to those pieces. Most of the program's work in any particular reformulation effort is in determining how that knowledge relates to the problem at hand. This involves the chunk setup, association, and incorporation processes discussed earlier. Since this process is controlled by the theme, trend, edge mechanism, it is necessary to see how this data structure is set up before the model application process can be discussed. In the next section, I will discuss how features of the modelled system are chosen for inclusion in themes, trends, and edges. In sections 3 and 4 we will see how those features are used to control the setting up and association of chunks.

Section 2 The modelled system

Consider some of the immediate questions which must be answered in order to reformulate a problem in terms of the basic idea of personnel flow mentioned above:

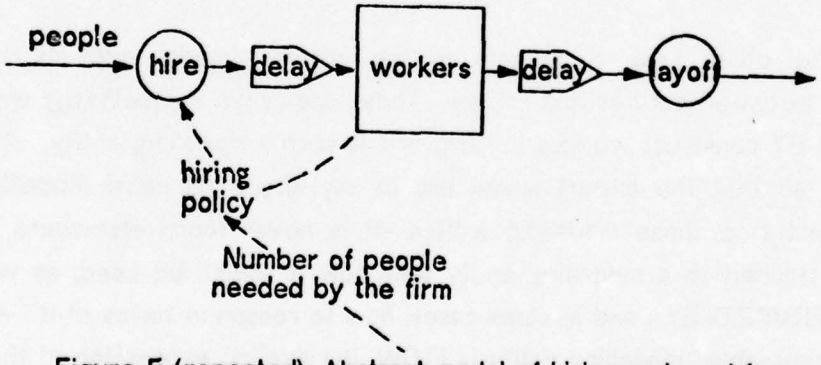


Figure 5 (repeated). Abstract model of hiring and workforce

-1- where do people come from?

-2- what is the delay after hiring?
- how long does it take?

- must it be there?

It is the job of the chunk setup mechanism to express questions like these in terms of pieces of the abstract model. That is, the chunk that is set up must be *a description of what is needed to bring a particular piece of abstact model into the modelled system (i.e., the solution so far).* This requires a little explanation.

Alfred works by associating chunks with pieces of the problem description and putting them into the modelled system. Now clearly, since the program can only solve problems in terms of the abstract model, these chunks must be based on the appropriate pieces of abstract model. However, it is the problem description which determines the details of the desired modelled system, as well as its "goal" in terms of the problem that must be solved. Therefore, in order for the prospective piece of the modelled system to "make sense" (i.e., be solvable) in terms of a given problem description, it must include the constraints and the needs of that problem description. So then, Alfred must set up chunks which consist of the required piece of abstract model, tailored to the existing constraints and overall goals of the problem description. It turns out that, for the program, this is equivalent to the neater formula I gave above: a description of what it takes to bring the piece of abstract model into the modelled system.

This is because as each chunk is associated, the new problem description information realized in its solution is added to the already existing base of "goals and constraints" in the modelled system. The modelled system thus always presents the current frontier of "everything you need to know to set up a good chunk". The modelled system conveys this knowledge to the setup mechanism via the special contextual information described earlier.

The format of this information is dependent on the way in which pieces are related to each other in the modelled system. Let me get at this relation between pieces by going back to our specific example. Suppose that Alfred, sometime during its processing of the case of figure 3, is trying to understand the hiring and layoff practices within the firm. It will need to reformulate the necessary information in terms of the basic idea of the labor sector. To do this, it starts by setting the basic idea we saw in section I-4. However, as I said above, to perform the association it must answer questions like

those I gave earlier. Now the important thing is that the program works on answering these questions in the same way it works on everything else: by setting up pieces of the abstract model and associating them. Thus, in order to associate the basic idea, it must set up and associate some more pieces.

A clearly defined pattern emerges: the association of a chunk sets off chain-reaction setting of pieces closely related to the chunk until some piece can be associated "directly", that is, without requiring another piece to be set up[1]. What this means in terms of the modelled system is that pieces become organized *during* the modelling process into a number of separate, highly intradependent "close interaction groups". Thus, every piece of the modelled system can be seen as being dependent on three (overlapping) sets of other pieces: its nearest "neighbors"--the pieces which directly caused it to be set and which it sets directly--(very heavy interdependence); its close interaction group (strong interdependence); and the entire set of associated and partially associated chunks--the modelled system--(much weaker interdependence).

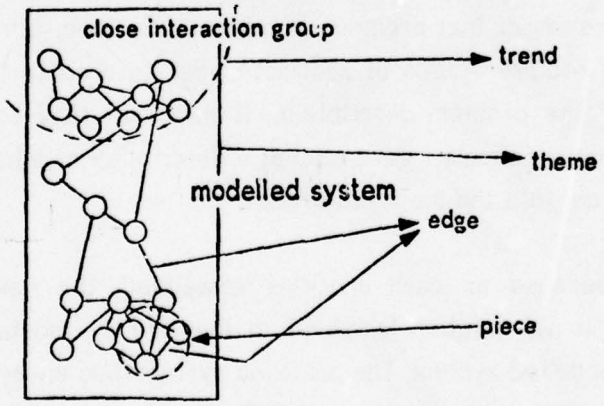


Figure 10. Themes, trends, and edges in the modelled system

For example, in order to model the workforce flow chunk given above, the

[1]The details of when an association can be done "directly" are determined by the abstracters attached to the chunk, the characteristics of the problem description, and, especially, the amount of problem-specific knowledge that went into setting up the chunk, as we will see in section 4.

program must figure out "where people come from"--i.e., associate the (PEOPLE) piece with some part of the problem description; "what the delay after hiring is", i.e., associate the DELAY1 piece; etc. The associations of all of the various pieces of the DETERMINED-BY construct form a "close interaction group". That is, the models of (HIRE) and (LAYOFF) are in the same close interaction group of the modelled system. The (HIRE) and (DELAY1) pieces are "neighbors"--they share an edge, the flow of people between them. Finally, (HIRE) is related to, say, the customer ordering activity, only in the weaker sense that both are part of the same reformulation effort. Models of sectors are the main close interaction groups in the modelled system. The relationship between the symptom and the problem definition of a theory forms another such group.

These three kinds of interdependency form the structural basis of the edges, trends, and themes which are maintained by the incorporation mechanism in order to convey needed modelled system information to the chunk set up mechanism. Edges are the more or less "syntactic" constraints on each piece due to its neighbors. As each piece of the abstract model is associated with a piece of the problem description, an edge is set up to convey this information to the next piece. The edge is that part of the previous association which the next association will have to deal with. For example, in our favorite workforce flow of figure 5, when the "hire" action is associated, the edge "employees" is set up, because the "delay" piece which is the next to be set up will have to deal with employees. The edge of an action within a flow is always the result of that action. The edge of an entity like "people" is either "people" itself, since that is what the hire action must deal with, or some more specific association like "ex-typesetters", if that is what is dictated by the problem description (because if "people" were restricted to "ex-typesetters", hire would deal only with ex-typesetters in that problem). Therefore, if the entire model is visualized in terms of flows like that of figure 5, the edge of one piece is always what the next piece "connects to".

Trends express the constraints on a piece that derive from its being in a particular close interaction group. This "semi-global" information is of four types. The first expresses the objective of modelling the close interaction group. For close interaction groups based on modelling sectors (i.e., almost all of them), this objective, called the TOWARD constraint, is the basic idea or other chunk that caused the sector to be entered (other TOWARD's come from the presenting symptom represented in the theme and the diagnosis of that symptom eventually found by the program's analysis routines). Thus,

when a decision is made to model the workforce sector of a firm, the basic idea flow of figure 5 becomes the TOWARD of the trend. For close interaction groups which arise from the effort of trying to find a theory which appropriately attacks a given symptom, the TOWARD is the symptom itself. We will see in the next section that the TOWARD is used in two ways: to limit the use of abstracters and to suggest focussing for a piece being set up.

The second kind of information in the trend is the PURPOSE constraint. It sometimes happens that in order to model a particular part of the problem, the program must go off on a sidetrack--to show that two items are similar enough to be considered the same, to show a contradiction, to look for the cause of some effect, etc. These are the "subgoals" which must be accomplished within the close interaction group. When the necessity of, say, finding a cause for effect x is seen, the PURPOSE constraint (SHOW-CAUSE x) is placed in the trend. This is a signal that the special cause-finding procedures attached to SHOW-CAUSE should temporarily take over the modelling effort, i.e., do their thing and depart. PURPOSE therefore serves the familiar subgoal function seen in many knowledge based programs.

One of the most important features of a close interaction group is that all of its component parts are to be modelled at the same level of detail. That is, if, for example, the production sector of the firm is to be modelled at a very high level of detail, all of its elements must be modelled at that same level of detail in order for things to "connect up" properly. This third kind of semi-global information is kept in the LEVEL property of the trend. This property is initialized with the actions and entities of the TOWARD constraint of the trend. As the modelling of the close interaction group proceeds, more and more actions and entities are added to the LEVEL property as they are seen to be at the same level as the actions and entities which are part of the objective of the trend. The decision as to whether or not something is at the same level as something else depends on its relationship with that thing via modelling entities. For example, all elements of a DETERMINED-BY are at the same level of detail. Level of detail propagation rules are based on the needs of the deductive knowledge attached to the modelling entities, and thus are an intrinsic part of the modelling entity as it is used by the system. For this reason, the modelling entity for expressing any given concept structure must be chosen with care. All of the modelling entities used in Alfred are described in Chapter V. The full rules for making level of detail decisions are given in Chapter VII.

The final kind of information that is put into the trend is related to the ABBREVIATION property. When a piece is associated at a higher level of detail than would normally have been the case, e.g., because the LEVEL property restricted the modelling knowledge attached to the chunk as we saw happen to DETERMINED-BY in the example of Chapter I, this fact is noted in the trend. In the case of the DETERMINED-BY seen earlier, the notation (ABBREVIATION (HIRE)) would be added to the trend when the hiring policy was verified. This is to indicate that this piece must be remodelled if the DETERMINED-BY structure is ever reexamined in more detail, and to show that the program can not yet make any inferences based on the hiring policy of Dominion.

Themes express the top-level goals of the modelling effort. They are always based on the symptom of the problem, and are of the form "find out what's wrong and fix it". Their importance is to signal the major phases of program operation like theory-finding and solution-suggesting. Also attached to the theme is the OPEN property. Entries under this property come from the ABBREVIATION property of the trend. If, in modelling a close interaction group, an entity from another sector, i.e., something that would have to be included in a different close interaction group, must be modelled, and if it is placed on the ABBREVIATION property of the trend (in other words, is not thoroughly modelled), it is entered on the OPEN property of the theme when the trend is closed. This is so the modelling mechanism will know that another close interaction group is "open"--i.e., must be dealt with in order to solve the problem.

Now that we have seen where theme, trend, and edge information comes from (there's a much fuller account in Chapter VII), let's see how it is used.

Section 3 Setting up pieces

The whole task of applying the abstract model to a particular problem description is made up of two basic activities: setting up pieces for association and associating them. The first activity is essentially the control of the problem-solving effort, that is the *selection*, *level of detail*, and *model tailoring* problems mentioned in Chapter I. In short, this is the activity of deciding which parts of the model are relevant to the problem, and at what level of detail they should be used at different phases of the problem-solving effort. The second activity is involved with making the association between the input and the abstract model piece once it has been set up. This comprises the *abstraction* and *focussing* problems discussed earlier. Both of these activities use the theme, trend, edge mechanism (as well as the knowledge attached to the entities, actions, and modelling entities of the abstract model), but in different ways. I will discuss the setup problem in this section and the association problem in the next.

A flowchart of the program's piece-setting-up activity is shown below:

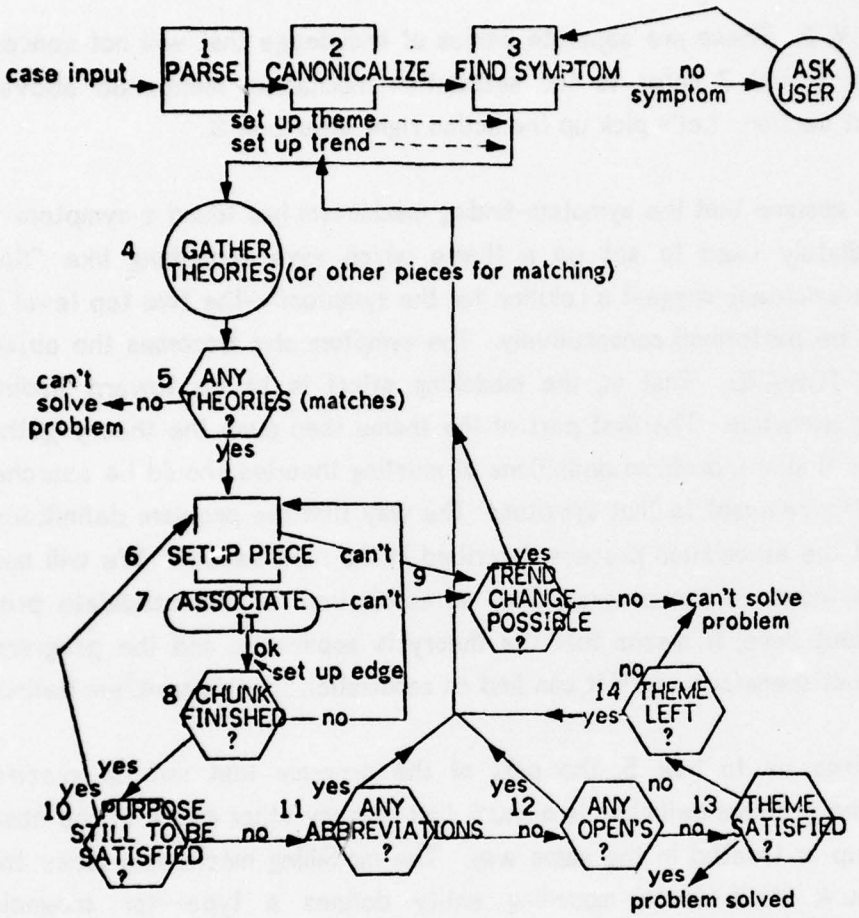


Figure 11. Flowchart of the setup effort

Boxes 1 and 2 are discussed in Chapter IV. The symptom-finding mechanism (box 3) is the

subject of section V-2. These are separate bodies of knowledge that will not concern us here. Also, boxes 4 and 7 refer to the association mechanism mentioned above and covered in the next section. Let's pick up the action right after box 3.

We will assume that the symptom-finding mechanism has found a symptom. This symptom is immediately used to set up a theme which says something like "find an explanation for the symptom; suggest a solution for the symptom"--the two top level goals of the program, to be performed consecutively. The symptom also becomes the objective of the trend, the TOWARD. That is, the modelling effort is to be toward finding an explanation for the symptom. The first part of the theme then cues the theory-gathering phase, which means that the problem definitions of existing theories should be searched to see if one is possibly relevant to that symptom. The way that the problem definitions are searched is part of the association process described in the next section. We will assume that a candidate is found. If the program finds an association for this candidate problem definition in the input case, it means that the theory is applicable, and the program can proceed. Alfred must therefore see if it can find an association for this problem definition.

This brings us to box 5, the part of the program that set up pieces for association. Now the problem definition is a chunk just like any other chunk of the abstract model, and its setup is treated in the same way. The modelling mechanism looks to see what type of chunk it is (each modelling entity defines a type--for example, a DETERMINED-BY chunk), and then looks at the modelling knowledge attached to that type. As discussed previously, the modelling knowledge will say what piece of the chunk to set up first. For example, the leftmost piece in a FLOW, or the third element of a DETERMINED-BY. If the suggested piece is also a modelling entity, the process recurses down and uses that modelling entity's modelling knowledge. At any rate, the modelling knowledge also specifies a default level of detail for the piece to be set up, usually the most detailed elaboration of whatever action or entity it happens to be. The suggested piece is then checked against the LEVEL property of the trend. If it is okay, it is set up. If it is not (i.e., in practice, if it is too detailed), the LEVEL property overrides the modelling knowledge of the chunk, and sets it at the right level of detail. This usually means picking a different elaboration of the piece, or simply verifying it. If there is no elaboration at the right level, the trend changing mechanism must be called in. That is, it is impossible to set up the piece, so something must be wrong with the modelling effort. Since the modelling effort is controlled by the trend, the trend will have to be changed if the effort is to continue. I will say more about trend changing in a minute.

For now, however, we will assume that it is possible to set up a piece at the right level, and that that piece is indeed set up. The program now tries to find an association for that piece, as discussed in the next section. If it can't make the association, something is again wrong, and the trend must be changed. If a successful association is made, the association is placed in the modelled system, with the edge of the newly associated piece being recorded as *the* edge for the modelling mechanism. The setup procedure then looks at the modelling knowledge of the chunk under consideration to see what piece should be set up next. Now, besides the level of detail constraint of the trend, it must look at the edge *from* the previous piece. This edge may require a different elaboration of a piece to be set up (say, if the suggested elaboration was an action that couldn't handle that edge as input), or, as we will see in Chapter VIII, may even cause a piece to be omitted--that is, edited out of the chunk. This is the way that edges are used for model tailoring. Indeed, they carry most of the burden for this task. The idea is that since the abstract model is coherent and self-consistent, the constraints of one piece on another should be of major influence in applying the model.

As long as the association of pieces is successful, and as long as the modelling effort is still working within a chunk, the modelling knowledge of the chunk is running the show. It can only be overridden by level of detail information. This is as it should be. The whole idea of the implementation methodology is to keep the knowledge pertaining to the modelling of a chunk or piece with that chunk or piece. Within the confines of a chunk, the modelling mechanism should assume that the chunk knows what it is doing. The only reason to override it is if there are more global reasons for doing things differently. In this case, the only possible influence is level of detail. (The way chunks are associated is also affected by the global needs of the problem, as we will see in the next section.) The edge can affect the local arrangement of a chunk, but can never exert a more global influence; that is, it can never override the modelling knowledge, it can only be used by it.

Once the chunk is completed, however, the program can no longer assume that local control is handling the modelling effort. It is time for more global needs to reassert themselves. The most local such global need--the smallest problem-solving unit that can go beyond a chunk boundary--is a PURPOSE constraint. The program therefore must check to see (box 10) if it is under a PURPOSE subgoal after it finishes the chunk. If the modelling mechanism is currently in a subgoal, it has given up control to the modelling knowledge attached to that subgoal, much as it gives up control to a chunk when it is in

that chunk (and subject to the same constraints from above). This subgoal may set up pieces in a non-chunk-oriented way to achieve its purpose. For example, the subgoal which checks for similarity between two pieces must set up and associate each piece one after the other, even though they may not be in the same chunk at all. The subgoal which finds causes for effects works by backtracing through flows, i.e., going in the opposite direction from the usual modelling effort. Therefore, the PURPOSE subgoals can be thought of special "chunks" which the program calls in when necessary to control a special purpose modelling effort. The PURPOSE subgoals are catalogued in Chapter VIII.

If the program has finished its current chunk or PURPOSE without a hitch, i.e., if all pieces that were set up could be associated and if the program was able to set up all the pieces it was supposed to (wasn't stopped by irresolvable level or edge conflicts), the trend is considered to have terminated normally. In this case, boxes 11 and 12 are used to pick a new trend. The idea is that the program has finished what it was supposed to in modelling the current interaction group or subgoal, and now is checking to see what is left for it. The most natural place for it to look first is under the ABBREVIATION property of the just finished trend (box 11). If anything is there, it means that the previous successfully modelled close interaction group is at a higher level of detail than is normally useful for solving problems. The usual reason for this is the "check through first and go into details later" approach described above. Therefore, if the program is under the "account for the symptom" part of the theme, it will assume that it should now go into details. It will therefore set up the first thing in the ABBREVIATION property as the objective (i.e., TOWARD) of a new trend, and proceed as before.

When there are no abbreviations left, the modelling mechanism checks under the OPEN property of the theme (box 12) to see if any other sectors need to be modelled in order to complete the problem. This works in very much the same way as looking under ABBREVIATION, except for one important difference. The idea of the OPEN mechanism is to provide a "mop-up" capability for the program--to allow it to finish up any sectors that are needed for completeness but don't have to be considered in detail. Therefore, when the trend is set up, the LEVEL property is automatically initialized with the most general actions and entities available, just as in a verification operation. In this way, the OPEN sectors will be modelled at the highest level of detail possible.

This has all assumed that the trend terminated normally, that is, that all pieces

in the close interaction group were properly set up and associated. If this is not the case, the modelling mechanism is on the wrong track, and the trend must be aborted. Basically, a new trend can be chosen in one of two ways here. Either a different elaboration of whatever was in the TOWARD of the failing trend must be found, or the LEVEL property must be changed. This is not always possible. For example, if the TOWARD was based on the symptom of the problem description, i.e., the initial trend, there is no alternate elaboration. Failure of the trend here means that the program was unable to recognize the symptom. It must therefore ask the user for a different statement of the symptom or simply give up. If a new trend cannot be found, it means that the program cannot get on the right track for the modelling effort, and must give up. The various possiblities for changing the trend are described in VII-5.

Assuming that the program is able to chug along setting and associating and is never at a loss to find a trend, it will reach box 13. Here it checks back with the theme to see if it has any major tasks left to perform. If all pieces are successfully set up and associated when it reaches box 13, which in practice means if it reaches box 13 at all, the "account for the symptom" part of the theme is considered to be satisfied--the program has found an appropriate theory and has modelled the problem to the extent needed by the theory. However, the whole theme has not been satisfied because the "find a solution" part has still not been satisfied. The program therefore moves on to box 14, where the "account for" part is taken off the theme. As this still leaves the "find a solution" part, the answer at box 14 is "yes", and the program begins work under the second part of the theme. The program operates just the same as before, setting up pieces, etc. until the solution has been found or it has to abort because of the lack of a trend. If things go well, the program will get down to box 13 again. The "find a solution" theme is considered to be satisfied if the user accepts the suggested solution. If he does, the answer at box 13 is "yes", and the problem is solved. If he does not, and if the program has no alternative solutions to offer, the "find a solution" theme is removed, and the program gives up at box 14. In either case, the problem-solving activity of the program stops at this point.

Note that this whole process relies on the "puzzle-setting" nature of the abstract model chunks to solve much of the selection problem. The flowchart of figure 11 shows no explicit mechanism for walking through the abstract model, or even for going from the theory-choosing phase to the problem modelling phase of program activity. Once the theory's problem definition is posited and the program's normal setup and association

process begins at box 6, Alfred assumes that the association of one chunk will naturally lead to the setup of the next, and so on until the problem is modelled enough for the solution finding process to be effective. The only deviations from this chaining process are the well-defined ABBREVIATION and OPEN cases, which are really more a matter of delaying steps in the chain rather than making major alterations in it. The way in which the association of one piece leads to the setting of another is clearly shown in the detailed example of Chapter VIII.

In the next section I will discuss the association activities represented by boxes 4 and 7 above. Here we will see the rest of the ways that trend information can influence the problem-solving effort.

Section 4 Associating pieces

Since, as discussed earlier, the same abstract model concept can be described in many different ways in the problem description, part of the program's task in its attempt to apply its model to a specific problem will be finding the correspondence between its model's abstract concepts and the concepts described in that problem. I have been calling this task "association". Now at some stage in this association process it is necessary to do "matching", that is, to say that problem description concept x can be taken as the representation of abstract concept y in this problem. Furthermore, since the same abstract concept can stand for a number of problem description concepts, this matching implies instantiation--x must be seen to instantiate or "be a kind of" y. This kind of matching by instantiation is often handled by reference to a type hierarchy [Winograd], [Martin] which is just a structure which says what can be a kind of what. In Alfred, all of the instantiation is handled by the abstracters which are attached to the actions and entities of the abstract model.

Within the abstracters attached to any action or entity are the problem description terms that that action or entity can legally be associated with. For example, attached to the abstract concept (REPETITIVE) is an abstracter which can attach it to the problem description concept (FREQUENT). The abstracters therefore form a sort of implicit type hierarchy, locally stored with each individual action or entity, in accordance with the

Alfred implementation methodology. If the abstracters were responsible for matching with every possible input concept that could come along, there would have to be an awful lot of abstracters. But this is not the case. The primary purpose of the canonicalization phase (box 2 of figure 11) is to reduce the number of legal associations for any given action or entity. The canonicalization phase thus represents a preselection or screening process which performs part of the instantiation effort by replacing terms in the problem description with standard synonyms before the abstracters even have to look at the problem. The instantiation activity of the canonicalization phase is limited to synonym substitution.

What then is left for the abstracters? That is, is there anything else in this matching business besides substituting a standard synonym for a group of similar concepts? The above case of (FREQUENT) and (REPETITIVE) indicates the beginning of the answer. Clearly, these two concepts are not synonyms. Neither is one "a kind of" the other. The canonicalization phase cannot simply substitute one for the other--this would involve a dangerous loss of information. As far as Alfred is concerned, (FREQUENT) and (REPETITIVE) are two concepts which can sometimes, but not always, be associated with each other. That is, (REPETITIVE) has some attached abstracters which allow (FREQUENT) as a valid match, and others that do not. For example, (REPETITIVE) is used to describe actions or entities which are known to be periodic or seasonal. In these cases, (FREQUENT) is not allowed as a match. On the other hand, something (REPETITIVE) which is known to happen PER (DAY) is allowed to be matched with the characteristic (FREQUENT). The decision as to which abstracters are allowed to be used when depends on the restriction mechanism of the trend which is described later in this section and again more fully in Chapter VII. That is, Alfred treats the fact that abstracters are only valid sometimes in the same way it treats the fact that the hiring policy of the firm can only sometimes be considered to be representable by the (HIRE) concept of the abstract model: the abstracter only represents the *possibility* of association, just as the elaboration only represents the *possibility* of a model. The problem of deciding the circumstances under which the association possibility represented by the abstracter or the modelling possibility represented by the elaboration can actually be used is left up to Alfred's general modelling procedure, as it is restricted by the theme, trend, edge mechanism which we will see in a second.

But this is not the only problem. As discussed in Chapter I, the manager's choice of concepts, their interrelationship, etc. may differ greatly from that of the abstract

model. Again, the canonicalization phase goes some of the way in that it standardizes the structure of the input and gives it the same *format* as the abstract model representation (i.e., it uses, wherever possible, the same modelling entities that are used to express the constructs of the abstract model--see Chapters IV and V). However, similarity of format does not imply similarity of concept structure: it only makes comparison easier. If the two conceptual structures are different (which they almost always are), it is necessary to alter the form of the abstract concept structure to make it conform more closely to the input concept structure. This is done by (1) choosing elaborations of the actions and entities in the abstract structure which are more similar to the input constructs, and (2) by rearranging the actions and entities in the abstract structure in order to make their interrelationship closer to that of the input constructs. As with the abstracters, the responsibility of choosing which transformations can actually be applied to abstract model pieces under consideration rests with the modelling mechanism.

The use of the abstracters should therefore be seen only as the terminal processing of a larger association effort. The rest of the process, i.e., anything that has to be done to bring a piece of the abstract model into "abstracter-matchable" range with a piece of the input, I have called focussing. That is, focussing is a process which is designed to transform the model chunk under consideration until it has the same structure as the input construct under consideration--until there is a concept-to-concept correspondence between the two structures. At this stage, the abstracters for each model concept in the transformed chunk can be run on the input construct to definitely ascertain whether or not a match exists. The control structure of the focussing and abstraction mechanism as well as its dependency on the rest of the modelling mechanism is shown in flowchart form below.

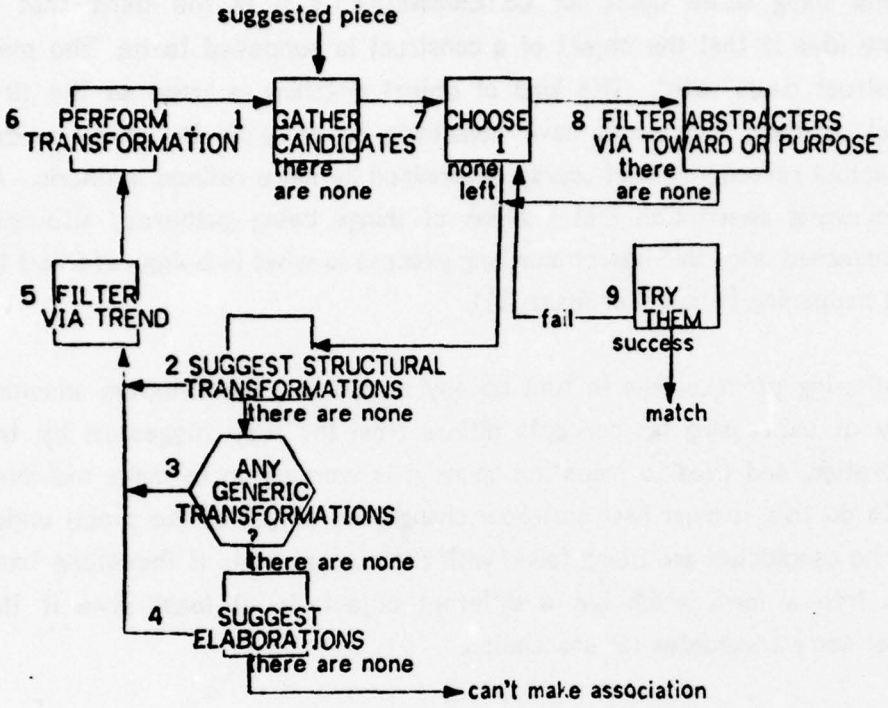


Figure 12. Flowchart of the association effort

The association process begins with the gathering of candidates that could possibly match the suggested piece of the abstract model. This is done by using the abstracters attached to the piece under consideration to fetch any construct of the input description whose **object** is abstracter-matchable to the object of the piece. The object is simply the first argument of the modelling entity which expresses the piece or construct. The gathering process therefore consists of finding all of those constructs whose objects the abstracters think are the same as the object of the piece under consideration. The representation scheme for the program (which we will see in Chapters IV and V) is so constructed that this first argument is always the "important thing" for matching purposes.

For actions it is the thing acted upon, for DETERMINED-BY's it is the thing that is determined, etc. The idea is that the object of a construct is supposed to be "the main thing that that construct deals with". This kind of object matching is used as the first rough cut to collect anything that could have something to do with the piece under consideration. The actual association is of course determined by more refined methods. At any point in the following description that I speak of things being gathered, alternate elaborations being collected, etc., this object matching process is what is being referred to. (This is also what is happening in box 4 of figure 11).

If the gathering process fails to turn up any candidates, the program assumes that the user's way of expressing his concepts differs from the way suggested by the piece under consideration, and tries to focus the chunk it is working on to make matching possible. In order to do this, it must first somehow change the object of the piece under consideration, since no candidates are being found with the current one. It therefore tries to focus the chunk into a form which has a different object, to at least give it the opportunity to collect some candidates for association.

The program has three ways of doing this kind of focussing. The first is to propose structural transformations: essentially to change the modelling entity which expresses the chunk under consideration to a different form. There are only a few such transformations, like changing

(DETERMINED-BY A (INFLUX B) (INFLUX C))

to

(FLOW B A C),

and changing structures like

(CAUSE-OF X Y)

to

(ACT-ON X ((CAUSE) MODIFICATION Y)).

These and a few other similar transformations are discussed in Chapter VI--we must wait to know more of the details of the representation scheme used in Alfred (Chapter V) before they can be explained more fully. Of course, only those transformations which change the object of the piece will be used.

If there are no such structural transforms, the program attempts to change the structure of **generics** in the chunk (box 3). Generics are special modelling entities like (CAUSE), DELAY, SMOOTH, etc. which represent a class or structure of abstract model actions and entities. For example, SMOOTH represents a time-averaging function; (CAUSE) can stand for a wide class of actions (essentially anything the firm does to a flow in the model). Generics will be discussd more fully in Chapter V. For our purposes here we can say that their use is similar to that of structural transformations: an expansion or substitution is offered which does not change the other actions and entities in the structure, but may change the way they are related to one another. The whole idea again is to translate the generic into something which changes the object of the piece under consideration so that association candidates can be found. For example, if a SMOOTH were in the object of the piece, "expanding" it into its functional form would change the object.

Finally, the program can suggest a different elaboration of the chunk (or part of the chunk) that changes the object structure (box 4). The elaborations are of course listed under the various pieces they pertain to. If there are no elaborations which can change the object of the piece, and there were none of the other kind of focussing transformations, i.e., the program drops through boxes 2, 3, and 4, Alfred cannot focus the chunk, and the association effort fails.

Assuming that a transformation was found in boxes 2, 3, or 4, the program must check to see if that transformation is in line with the modelling effort (box 5). The decision as to what transformations can be validly used to associate a piece of abstract model with an input construct depends on the objective of modelling that piece--it can't just blindly use an elaboration or other focussing transformation because it is available and still hope to make progress on the actual modelling effort at hand. For example, if the modelling effort were trying to show how hiring is affected by backlog, it would be inappropriate to transform the hiring chunk into something that showed how it was affected

by the possibility of union troubles, even though that is a valid elaboration of hire and is useful under some circumstances. As another example, if the program were trying to model the effects of a change in hiring on the rest of the workforce flow (a la figure 5), it would be inappropriate to consider the hiring policy as a numerical function--it wouldn't tell the program anything it needed to know. Of course, for the final simulation of the model, it is crucial to know the exact number of people hired at each time t. In terms of the Alfred implementation methodology, all of this means that the choice of focussing transformations must take into account the objective of modelling the current close interaction group--that is the TOWARD (or the PURPOSE when it has control of the modelling effort) of the current trend.

This "taking into account" takes the form of filtering the transformations on the basis of more restrictive elements in the TOWARD or PURPOSE. The rule is that if, in comparing the abstract model piece under consideration to the pieces of the TOWARD or PURPOSE, a more restrictive version of that same piece is found in the TOWARD or PURPOSE, only the elaborations of that more restrictive piece can be used. Similarly, if a structural transformation or a change in the structure of a generic is used, it must not introduce actions or entities which are less restrictive than those of the TOWARD. What does more restrictive mean? We again have to wait until Chapter V to get the full definition, but, simply put, entities like (PEOPLE (EMPLOYED (WORKING))) are more restrictive than (PEOPLE (EMPLOYED)), which is more restrictive than (PEOPLE), the action (HIRE) is more restrictive than, say, the generic (CHANGE); and, an elaboration of action or entity x is more restrictive than a piece containing x. By allowing the use of elaborations and structural transformations to introduce no actions or entities which are more general (less restrictive) than those specified in the TOWARD or PURPOSE, the program ensures that only forms of the piece which are within the scope of the actions and entities specifically mentioned in the TOWARD or PURPOSE can be set up. This is extremely important, as we will see in a second, because this is what restricts the scope of the abstracters that can be used to make associations. For example, if the TOWARD was dealing with "employees" (i.e., (PEOPLE (EMPLOYED))), and the piece under consideration were allowed to be focussed in such a way that (PEOPLE) could be set up, the abstracters under (PEOPLE) could make an association with, say, ex-typesetters, which is clearly moving away from the objective of modelling the close interaction group, as expressed in the TOWARD. Therefore, this scoping by filtering possible focussing transformations through the trend is what keeps the program on track during the modelling effort.

In addition to the above use of the trend to filter the possible transformations, any transformation on the piece under consideration must also comply with the level of detail restrictions of the LEVEL property of the current trend.

If a transformation can be found which conforms to the restrictions of the trend and changes the piece under consideration in a way which is significant to the gathering process, it is applied to the piece under consideration (box 6). The piece is then reset in its newly focussed state, and is set up. The gathering process then begins again for the piece.

Assuming now that the gathering process finds some candidates, the first of the "gathered" input constructs is chosen to be the input construct under consideration (box 7). Now the abstracters of the newly focussed abstract model piece can be applied to the input construct under consideration to see if there is a match. But, before these abstracters can be applied, they too, like the focussing transformations suggested in boxes 2, 3, and 4, must take into account the needs of the modelling mechanism.

Just as the appropriateness of using available elaborations and transformations of a chunk to focus it depends on circumstances, the appropriateness of using available abstracters to match it depends on the needs of the modelling effort. Going back to the previous example, if the program were trying to model the effects of backlog on hiring, it would be inappropriate to use an abstracter which could associate workers with union members or non-union members. Again, it's not that this association is wrong, it's only that it is inappropriate under those circumstances.

Therefore, as mentioned above, before the abstracters are used, they are filtered against the TOWARD or PURPOSE of the trend much in the same manner as the suggested transformations. The rule now is that if the TOWARD or PURPOSE contains an action or entity which is more restrictive (in the same sense as before) than the piece under consideration, only the abstracters of the more restrictive action or entity can be used. In this way, the modelling mechanism is assured that only associations which are relevant to the needs of the modelling effort (at that time) can be made. (Note: the reason why this filtering must be done "again"--i.e., the abstracters must be explicitly filtered even though the pieces they are taken from have already been filtered--is that, looking at the flowchart, it is possible to reach box 8 without going through the previous filtering

process: the piece under consideration may not have been focussed before. Furthermore, since focussing is only applied to some parts of the piece under consideration, it is possible that the part whose abstracters are being used was not one of those that was focussed previously.)

If an abstracter of the piece under consideration is okay with respect to the restrictions of the TOWARD or PURPOSE, it is tried (box 9). If it finds a successful match on one of the gathered input constructs, the association effort on the piece is successfully completed. If after going through all of the input candidates, no match is found, the association effort checks to see if there are any other ways to focus the chunk. If not, the association effort fails.

There is only one case left to consider. If the gathering effort is able to find candidates (i.e., it gets to box 8), but none of the abstracters are able to pass the filter of the TOWARD or PURPOSE, the focussing mechanism is again called into play (via the "there are none" exit of box 8). But this time there is an important difference. Now there is a definite input candidate which can be checked against the piece under consideration to find specific differences between the two structures which the focussing mechanism can try to eliminate.

A piecewise comparison of the two structures is instituted. If the two structures are not of the same type, i.e., are expressed by different modelling entities, there is a possibility that a structural transformation can be performed to express the two structures in terms of the same modelling entity. This time only structural transformations which can map the modelling entity of the model piece into the modelling entity of the target input construct are considered. The mapping of generics into the structures they represent is handled in a similar way. For example, the generic action (CHANGE) can be mapped into almost any action that occurs in a flow. Also, the generic SMOOTH can be mapped into the time-averaging structure that it represents, if the input construct seems to be describing such a structure. If there are no transformations of this type that would accomplish the purpose of making the piece structure look more like the input structure, alternate elaborations are investigated. An elaboration of a part of the piece under consideration (the largest part is examined first, then recursively smaller ones) is chosen if its modelling entity is the same as that of the corresponding part of the input structure. This elaboration then becomes the new piece under consideration (after it passes trend

filtering). The association process, and, if necessary, the whole focussing process is then repeated on the new piece. Therefore, if there is a target input piece in mind, it can be used to direct the focussing efforts of the program. Any transformations chosen on the basis of differences between the chunk and the input structure under consideration must of course also be filtered through the trend, exactly as before.

This then is the association procedure of the program, applied recursively on each part of the model piece under consideration whenever a match between it and some piece of the problem description is desired. We will see the representation of the manager's input description in Chapter IV and the abstract model representation in Chapter V. After these representations are known, Chapters VI and VII can go into more details about how focussing transformations and trend filtering works.

Section 5 Diagnosis and solution

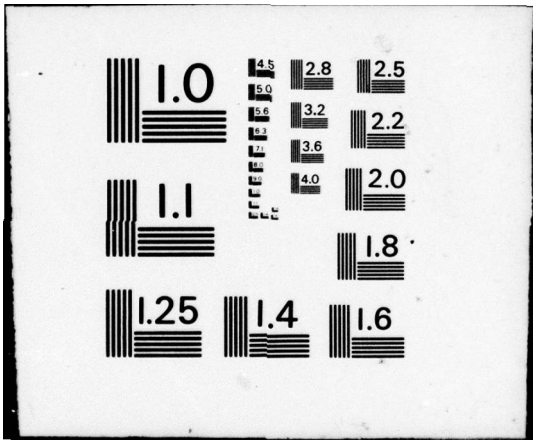
Once Alfred has finished modelling the problem situation (i.e., has created a complete modelled system), it must "solve" the manager's problem by suggesting some sort of change to eliminate or alleviate the undesirable characteristics of the firm. This suggestion usually takes the form of a policy change to be implemented by the firm. As I mentioned above, these prescription policies are attached to individual theories. The problem is to determine which of several possible prescriptions to choose and how to apply it to the specific problem at hand.

It is hardly surprising that this sounds exactly like the kind of modelling we have been discussing in connection with the rest of the abstract model. Indeed, policy choice and application is handled by the same modelling mechanism after the modelled system has been finished (if indicated by the theme; see Chapter VII). Although this particular association effort can sometimes be rather involved (perhaps requiring simulation of the model to look for a particular dynamic characteristic[1]), it is conceptually easy for the program (i.e., it doesn't involve much "deep" reformulation reasoning). This is because the proper foundation for policy implementation has been laid by the earlier modelling effort.

[1]The modelling scheme used in the abstract model completely defines a set of simulation equations for the system, so the modelled system is directly simulatable.

AD-A035 397 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
THE REFORMULATION MODEL OF EXPERTISE.(U)
DEC 76 W S MARK
UNCLASSIFIED MIT/LCS/TR-172 N00014-75-C-0661
NL

2 OF 3
ADA035397


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

Also, since the system has been completely modelled, the association effort here becomes mostly a matter of looking at known features of the problem (and their **reasons**--see Chapter IV) to see if they *prohibit* a particular policy recommendation: i.e., it is often the case that only negative checking is required.

Policy alternatives and their criteria for applicability will be discussed in V-1.3. A suggested policy change which "solves" the problem of the Dominion case shown earlier will be described in Chapter VIII.

The following chapters provide a more thoroughgoing explanation of the program elements described here. Chapter VIII gives a detailed look at Alfred's problem-solving effort for the Dominion case in order to illustrate the use of these program elements.

Chapter IV
The Input

In most real life consulting situations, the problem is described and the consultant's advice is given in terms of a natural language dialogue. Unfortunately, the difficulties of conducting even the simplest dialogue are so formidable [Brown] that natural language dialogue is out of the question as a vehicle for Alfred. Furthermore, as is well known, natural language processing is not in a state which would allow a program like Alfred the free use of natural language, even in a non-dialogue setting. Nonetheless, I want it to be completely clear at every point of the problem-solving effort what kind of input information Alfred is processing. For this reason, I have reached the following compromise.

Input to Alfred is in English, the freedom of expression determined by the limits of the parser and other natural language routines I have written. Furthermore, although the user can ask questions and give answers, the bulk of the input information is given in the form of a "brief"--several paragraphs describing the firm and its problems. As a final approximation, instead of using real managers as users, I use a business "case" for the brief. A case is a business problem situation, almost always taken from a real life consulting experience, which has been isolated and written up (usually for the purposes of teaching). Since cases incorporate most of the important characteristics of real life problem descriptions (extraneous information, unstructured input of unknown order, no explicit definition of symptoms, etc.), I don't think that there is significant loss of generality in using them. A couple of paragraphs (right out of the book [Jarmain]) from the description of the "Future Electronics Company" will give the flavor of a case (figure 13). The simpler English form into which this description must be converted in order to actually be input to Alfred is given in figure 14.

"Simpler" English means several things. First of all, sentence structure is made more uniform to ease the burden on the parser. In some cases, synonyms are substituted for words or constructs so that Alfred's vocabulary can be smaller. Also, policies mentioned in the case are explicitly designated and named. This is due to the fact that policies are

the consultant's "working variables". That is, since policies are often the cause of the problem, and since they are the most readily changeable aspects of the system, the consultant's "solutions" to system problems will usually be in the form of recommendations for policy change. The policies must therefore be easy for both the manager and the consultant to talk about. Finally, it sometimes happens that real world knowledge is implicitly contained in the structure of a sentence or in the juxtaposition of two sentences in the case. It is then necessary to put that information into the simplified sentences that convey that information. An example of this is the sentence about disruption of production by poor quality shipments. Alfred has no way of connecting the fact that disruption of production causes buyers to choose a different supplier (a fact conveyed by the juxtaposisition of two clauses, and the world knowledge that "more dependable quality" will lead to fewer disruptions). The information must therefore be given explicitly in its version of the case. This kind of thing doesn't come up too often (primarily because the cases are rather simply written in the first place).

The simplified input is then parsed into the format shown in figure 14, which can be taken as a representation of the level at which *anybody* would understand the problem description. It is primarily a case-based representation (i.e., things can have agents, sources, destinations, durations, etc.) centered around action/object pairs. Both verbs and nouns can be modified. A list of the modifiers appears after the noun or verb; lack of modification is explicitly represented by NIL. The parsed structure is then placed under one of the "subjects" recognized by the parser: FIRM, CUSTOMER, COMPETITOR, or USER. It is actually added to the value of a property based on the subject of the sentence; i.e., if the sentence is talking about the firm's shipments, it is placed on the SHIPMENT property of FIRM. If the subject is the firm (customer, competitor, etc.) itself, the sentence is placed under the CHARACTERISTIC property of that subject. If-then sentences are placed under unique CAUSE-EFFECT properties of the appropriate subject (they are parsed into a condition/result format).

After the entire case has been parsed in this manner, the program goes a step further, translating the parse into the "abstract language" representation of figure 15, its interpretation of the way in which the *consultant* would remember the description. Note that this does not imply that any reformulation, or indeed that any sophisticated processing of any kind, has occured. Alfred has just put the description into the slightly more formal version it uses for talking to itself. That is, canonical synonyms are substituted for many

words (especially verbs); hyphenated noun structures (which come from series of classifiers) are broken into their component parts; action/object and condition/result structures and case properties are made more uniform, NIL's are removed (NIL's are also put in during the canonicalization phase, but here they mean "unspecified", not "lack of modification"). Finally, the canonicalized information is placed under the MANAGER-DESCRIPTION property of an abstract model entity. The appropriate entity is just the canonicalized version of the property (i.e., like SHIPMENT) under which the sentence was previously stored. Definitions and format information concerning abstract model entities are given in the next chapter. Sometimes information is lost in this canonicalization process. However, it is well worth the information loss to allow the program a standard problem description environment to work in.

A complete case was shown in figure 3. Its translation into Alfred input English is given in the detailed example of Chapter VIII.

The parser and canonicalization phase also accept the user's answers to program questions. After the program has enough information to get a general picture of the manager's organization, it can begin the reformulation process. During that process, however, it will need to elicit additional information from the manager--to fill out part of the problem description, to confirm a hypothesis, etc. Alfred must therefore provide a reasonable question-and-answer environment for allowing this interaction. Answers are just added to the problem description as if they had been input in the case. The questions are generated by the modelling process, as we will see later.

The management of Future has for some time been worried about the image of their quality held by their customers. From time to time they have heard statements made by their customers such as "We are generally quite satisfied with your quality, and consider you one of our highest quality suppliers, but are bothered by some of the variations which occur. Every so often, we receive a series of poor shipments from you. These create a disruption of our production, and we may be forced to find a supplier whose quality is more dependable, even if their best is not as good as yours." While customers are not always as outspoken as this, Future has noticed that there are periods of time when many defective units are returned, but at other times very few defectives are returned.

The management of Future is quite sensitive to this situation, and when they notice that complaints and returns are increasing, they hire more people in order to increase the thoroughness of the testing procedure. They base their hiring decision on the number of testers presently employed and on the frequency of complaints.

Figure 13. Part of the Future Electronics case (from the book)

Future has been worried during recent periods about the customer's image of product quality. Customers are generally satisfied with product quality. Customers consider that Future supplies products of the highest quality. The customers' occasional complaint is that sometimes product quality is bad (the quality of series of Future product shipments is sometimes poor). Poor shipments disrupt customer production. If competitor product shipment quality is more dependable, customers will buy from competitors, because the customer's production will not be disrupted by the competitor's shipments. Many defective units are returned during some periods. Usually, few defective units are returned.

Future hiring policy is: if customer complaints and product returns are increased, hire more people (increase the thoroughness of quality control). The hiring decision considers the number of "tester"s presently employed and the frequency of customer complaints.

Figure 14. Same part of the Future case (actual input form)

```
FIRM:

CHARACTERISTIC  (PAST-STATE: (WORRIED)
                 DURATION: (PERIOD (RECENT))
                 REASON: (CUSTOMER-PRODUCT-QUALITY-IMAGE NIL))

SHIPMENT  ((POOR) (ACTION: ((DISRUPT) NIL)
                   OBJECT: (CUSTOMER-PRODUCTION NIL)))

CAUSE-EFFECT-1  (CONDITION: (STATE: (COMPETITOR-PRODUCT-SHIPMENT-QUALITY NIL)
                             (DEPENDABLE (MORE)))
                 RESULT: (ACTION: ((BUY) NIL)
                          OBJECT: NIL
                          AGENT: (CUSTOMER NIL)
                          SOURCE: (COMPETITOR NIL))
                 REASON: (FUTURE-STATE: (CUSTOMER-PRODUCTION NIL)
                                        (DISRUPTED (NOT))
                          AGENT: (COMPETITOR-SHIPMENT NIL)))

UNIT  (((MANY DEFECTIVE) VALUE: (RETURNED)
        DURATION: (PERIOD (SOME)))
       ((FEW DEFECTIVE) VALUE: (USUALLY RETURNED)))
```

Figure 15. The parse of that part of the Future case

```
HIRING-POLICY   (CONDITION: (STATE: (AND (CUSTOMER-COMPLAINT NIL)
                                         (PRODUCT-RETURN NIL))
                                    (INCREASED))
                RESULT: (ACTION: ((HIRE) NIL)
                         OBJECT: (PEOPLE MORE))
                    REASON: (ACTION: ((INCREASE) NIL)
                             OBJECT: (QUALITY-CONTROL-THOROUGHNESS NIL)))


HIRING-DECISION   (ACTION: ((CONSIDER) NIL)
                   OBJECT: (AND (TESTER-NUMBER (PRESENTLY EMPLOYED))
                                (CUSTOMER-COMPLAINT-FREQUENCY)))


CUSTOMER:
 CHARACTERISTIC   ((STATE: (PRODUCT-QUALITY NIL)
                           (GENERALLY SATISFIED))
                  (ACTION: ((CONSIDER) NIL)
                   OBJECT: ((ACTION: ((SUPPLY) NIL)
                             AGENT: (FUTURE ELECTRONICS INC))
                            OBJECT: (PRODUCT-QUALITY (HIGHEST)))))


COMPLAINT   ((OCCASIONAL) (STATE: (PRODUCT-QUALITY NIL)
                                  (SOMETIMES BAD)
                            REASON: (STATE: (FUTURE-PRODUCT-SHIPMENT-SERIES-QUALITY NIL)
                                     (POOR))))
```

Figure 15. (cont.) The parse of that part of the Future case

```
((FIRM) (CHARACTERISTIC))
    (MANAGER-DESCRIPTION
      ((STATE-OF (FUTURE ELECTRONICS INC)
            (CONCERNED))
        TIME (IN-PAST)
        DURATION ((PERIOD) MODIFICATION (RECENT))
        REASON ((QUALITY-PERCEPTION) OF (CUSTOMER))))


((FIRM) (PRODUCT (GROUPED)))
    (MANAGER-DESCRIPTION
      (ACT-ON (STATE-OF ((PRODUCTION) OF (CUSTOMER))
                  (UNMANAGEABLE))
          (CAUSE)))


((FIRM) (CAUSE-EFFECT) (1))
    (MANAGER-DESCRIPTION
      ((DEPENDS-ON (ACT-ON ((PRODUCT) MODIFICATION (ASSUMED))
                       ((PURCHASE)
                         AGENT (CUSTOMER)
                         SOURCE (COMPETITOR)))
                 (STATE-OF (QUALITY-OF (NUMBER-OF (PRODUCT (GROUPED)) (INCREASED)))
                       (CONSTANT)))
          REASON (STATE-OF ((PRODUCTION) OF (CUSTOMER))
                     (UNMANAGEABLE))
                 AGENT ((PRODUCT (GROUPED)) OF (COMPETITOR))))


((FIRM) (PRODUCT))
    (MANAGER-DESCRIPTION
      ((CHARACTERISTIC (NUMBER-OF (PRODUCT (RETURNED)) (MANY))
          REASON (STATE-OF (PRODUCT (RETURNED))
                           (DEFECTIVE))
          DURATION ((PERIOD) MODIFICATION (SOME)))
       (CHARACTERISTIC (NUMBER-OF (PRODUCT (RETURNED)) (FEW))
          REASON (DEFECTIVE)
          MODIFICATION (USUALLY))))
```

Figure 16. The canonicalization of the parse

```
((FIRM) (POLICY) (HIRE))
    (MANAGER-DESCRIPTION
      ((DEPENDS-ON (ACT-ON (NUMBER-OF (PEOPLE) (INCREASED))
                           (HIRE))
                   (STATE-OF (AND ((COMPLAINT) OF (CUSTOMER)) (PRODUCT (RETURNED)))
                             (INCREASED)))
           REASON (ACT-ON (EFFICIENCY-OF (QUALITY-CONTROL))
                          (INCREASE))))

((FIRM) (HIRE))
    (MANAGER-DESCRIPTION
      (ACT-ON (AND ((NUMBER-OF (TESTER (EMPLOYED)) NIL)
                           MODIFICATION (CURRENT))
                   ((NUMBER-OF (COMPLAINT) NIL)
                           OF (CUSTOMER)
                           PER (PERIOD)))
              (CONSIDER)))

((CUSTOMER) (CHARACTERISTIC))
    (MANAGER-DESCRIPTION
      ((STATE-OF (QUALITY-OF (PRODUCT))
                 (SATISFACTORY))
           MODIFICATION (USUALLY)))

((CUSTOMER) (CHARACTERISTIC))
    (MANAGER-DESCRIPTION
      (ACT-ON ((ACT-ON ((QUALITY-OF (PRODUCT)) MODIFICATION (HIGH))
                       (PRODUCE))
               AGENT (FUTURE ELECTRONICS INC))
              (CONSIDER)))

((CUSTOMER) (COMPLAINT))
    (MANAGER-DESCRIPTION
      ((SOMETIMES)
        ((STATE-OF (QUALITY-OF (PRODUCT))
                   ((BAD) MODIFICATION (SOMETIMES)))
             REASON (STATE-OF (QUALITY-OF ((PRODUCT (GROUPED (GROUPED)))
                                           OF (FUTURE ELECTONICS INC)))
                              (BAD)))))
```

Figure 16. (cont.) The canonicalization of the parse

Chapter V
The Knowledge Base

The heart of any expert program is its knowledge base. This is what ultimately determines the range of the problems the program can attack and establishes the bounds of its applicability. For Alfred, the knowledge base is more than just the abstract model (though this is the major part). The applicability of the program is also determined by the symptom-finding mechanism, the range of the abstracters, and the contents of the modelled system. In this chapter I will discuss each of these elements of Alfred's knowledge base. Chapters VII and VIII are devoted to showing how the various parts of the knowledge base are used by the program to solve problems.

Section 1 The abstract model

Any model is a representation of something. If a model is to be used as part of a knowledge base, it must also contain knowledge *about* the thing it represents. Some of the knowledge is implicit in the representation itself, via the choice of modelling constructs, the way the model is stored and accessed. That is, the program expresses a certain amount of its expertise about the domain simply in the choice of representation: some kinds of relationships between concepts are made easy to express, others turn out to be harder. In Alfred, for example, flow relationships are easy to express while, say, procedural relationships are much harder, thus showing Alfred's bias for flow modelling. Combined with this implicit knowledge is the explicit knowledge which is associated with the representation: the interpretation of individual modelling constructs, constraints on the way constructs can be combined, and domain-dependent knowledge about the entities represented by the modelling scheme. Explicit knowledge gives the properties of the various elements of the representation scheme along with procedural knowledge about how to use these elements to solve problems.

Consider, for example, the representation and implementation of a function in a

programming language like LISP, say, (PLUS A B). There is some knowledge implicit in the representation itself: the functional form is easy to parse and easy to use recursively. On the other hand, it is somewhat unfamiliar and thus initially hard to use for some people. This implicit knowledge already says a great deal about how LISP can and should be used. The explicit knowledge attached to PLUS says that it is a function and gives the algorithm for computing the sum of its arguments. Together, this implicit and explicit knowledge makes up everything LISP has to say about PLUS. Similarly, implicit and explicit knowledge combine to form the model's contribution to the knowledge base of Alfred. I will discuss each kind of knowledge separately below. Before going into this, though, I will describe the "style" or point of view embodied in the way the implicit and explicit knowledge of the model is conceived and used.

Alfred is biassed toward a cause-effect view of flows and actions within a system. This is a very common (expert) way of looking at both physical and social systems. The organization is viewed essentially as an interconnection of processors which act on the entities of interest in accordance with the physical laws or policies of the system. The interconnection is in the form of *flows*, not only of physical entities, but also of information about those entities. This information is what is used by the laws or policies to govern the flow of entities (and the flow of the information itself) within the system. The theory is that by proper consideration of *both* of these kinds of flows, and knowledge of the laws and policies of the system, the cause of every effect in the system can be determined. In particular, in Alfred's role as consultant, the model will be used to find the cause of undesired effects, and then to find a way to cause the desired effects.

An example of a cause-effect flow model from Alfred's working domain of business systems will help to clarify these concepts. Let's begin with a simplified version of the flow model of how people are employed by the firm which was discussed in Chapter I:

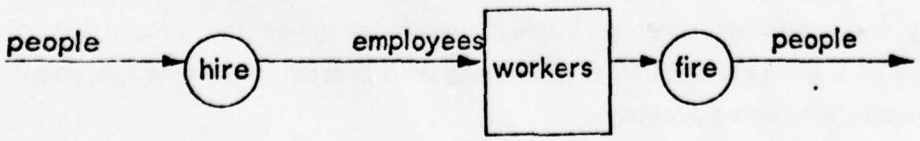


Figure 17. A basic flow notion of employment

"People" flow into "hire" where they are turned into "employees", and stay employees until they flow out of "fire". The box labelled "workers" represents the accumulation of employees at any given time. Of course, this model doesn't tell us very much. The usual questions of interest about this part of a firm deal with when, why, and how people are hired and fired. This is where information flows enter the picture. Presumably, the hiring decision is based on how many employees are currently in the firm, the number of workers. The firing decision may also depend on this. Both the hiring and firing policies may also use the number of people available for employment as a decision variable; the number of employees needed by the firm to do its work is also an important consideration. The picture has become rather more complicated:

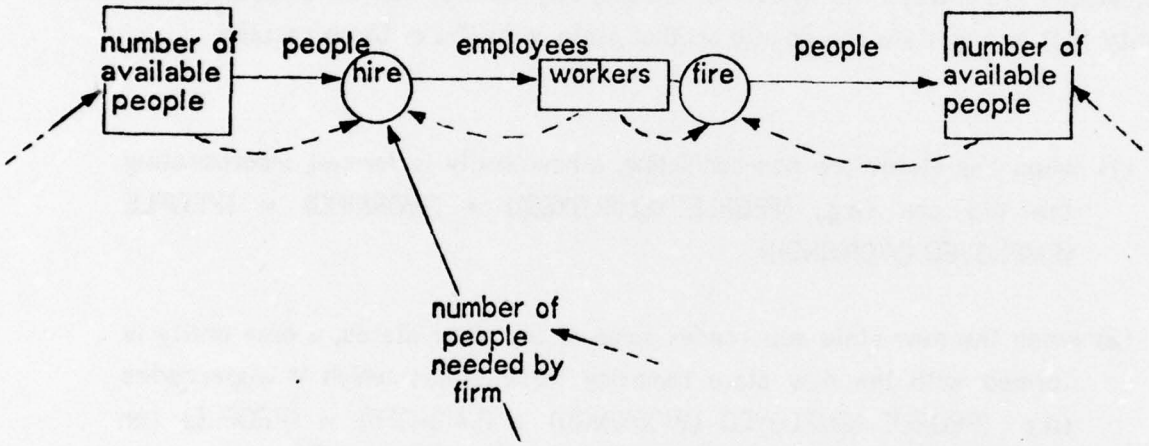


Figure 18. A more revealing flow notion of employment

Furthermore, since the "number of people available" and "the number of people needed by the firm" are both determined by factors outside of the model shown so far, the model must expand still further if we are to really answer those questions about hiring and firing. But the faith is that these factors too can be represented in terms of flows and actions. What Alfred is really good at is deciding just what in the firm has to be modelled in order to answer the questions of interest, and then how to determine the causes of the relevant effects (which is really what the questions are asking) once enough of the firm has been modelled. But before it can do any of that, it needs a representation for all of its knowledge about firms and flows.

1.1 Representing the abstract model

The basic unit of the abstract model is the **entity**. "People" in the above example would be represented by the *atom* (PEOPLE), an entity in Alfred's abstract model. Now, "employee" could also be represented by an entity (EMPLOYEE), however, in so doing the relationship between people and employees (via hiring) would be obscured. Also, the abstract model would end up having to worry about a great many entities. Therefore, concepts like employee are represented by the entity (PEOPLE (EMPLOYED)) (again an atom). (EMPLOYED) is a **state**, the result of the **action** (HIRE). Only entities can be in states, states are always the results of actions, only entities can be affected by actions. An entity that is one state can go into another state, with one of three results:

(1) when the states are non-conflicting, a new entity is formed, incorporating the old one (e.g., (PEOPLE (EMPLOYED)) + (WORKING) = (PEOPLE (EMPLOYED (WORKING))),

(2) when the new state supercedes some of the other states, a new entity is formed with the new state removing those states which it supercedes (e.g., (PEOPLE (EMPLOYED (WORKING))) + (LAID-OFF) = (PEOPLE) (or (PEOPLE (LAID-OFF)), if that is an entity of interest),

(3) when the new state is contradictory with respect to some of the other states, an error has occurred (e.g., (PEOPLE (EMPLOYED)) + (BACKLOGGED) = error).

(This third condition never arises in the current Alfred program--the action that results in the (BACKLOGGED) state would never be applied to (PEOPLE), etc. I include the circumstance for generality.) Entities and actions can also be particularized by other entities, as in ((CUSTOMER) (ORDER)) or ((FIRM) (POLICY)).

Entities are what make up the "flows" discussed earlier; actions are the "processors" which work on these flows. The modelling process expects entities and actions to be associated with pieces of the problem description. Of course, as

overemphasized earlier, these associations are by no means direct, the way the model uses these concepts is quite different from the way the manager uses them, etc., etc. However, we can certainly say that (HIRE), for example, will be associated *in some sense*, with the hiring policy of the firm in question, (PEOPLE (EMPLOYED (WORKING))) will be associated with the workers of the firm, etc.

On the other hand, there are some parts of the model which are in no sense to be associated with pieces of the problem description. These are the **modelling entities**, the model's devices for showing relationships between entities: they need have no direct correspondence with a piece of the problem description, but are the mechanisms of the model to express things in the model. In the LISP PLUS function shown earlier, "PLUS" would correspond to the modelling entity, and "A" and "B" to actions or entities. That is, "A" and "B" are to be associated with pieces of the input, while "PLUS" is that part of the LISP system which structures and expresses the "modelling" operation of summing. This analogy is a little dangerous, mostly because actions and entites are not really very much like variables in a programming language, but it does make the distinction between modelling entities and regular entities and actions clear. Alfred has modelling entities to express activities, dependency, and functional relationships.

For example, the processing of an entity by an action is represented by the modelling entity ACT-ON, as in

```
(ACT-ON (PEOPLE)
        (HIRE)).
```

Similarly, the fact that an action or entity depends on another action or entity is expressed by DEPENDS-ON, as in

```
(DEPENDS-ON (WORKLOAD)
            ((CUSTOMER) (ORDER))).
```

The most common use of DEPENDS-ON is in conjunction with another modelling

entity, STATE-OF, which is used for two purposes: to indicate that it is the **state** of an entity that is being referred to, not the whole entity; and to express the fact that entity A is in state B when (A (B)) is not an entity that the system knows about (e.g., Alfred knows of no entity (PEOPLE (EMPLOYED (IDLE))), though it may have to deal with employees in the idle state)[1]. Thus,

```
(DEPENDS-ON
   (STATE-OF (PEOPLE (EMPLOYED (WORKING)))
             (IDLE))
   (STATE-OF (WORKLOAD)
             (INSUFFICIENT)))
```

means that the fact that (PEOPLE (EMPLOYED (WORKING))) are (IDLE) is due (at least in part--there is no claim of sole dependency) to the fact that (WORKLOAD) is (INSUFFICIENT).

There are selector functions analogous to STATE-OF for some of the most commonly used generic characteristics. That is,

```
(EFFICIENCY-OF (PEOPLE (EMPLOYED)))
```

means that it is the employees' *efficiency*, not the employees themselves, that are being referred to. The other selectors used in Alfred are QUALITY-OF and CAUSE-OF.

Quantitative entities, i.e., entities that can be in numerical "states" are described by NUMBER-OF, as in

```
(NUMBER-OF (PEOPLE (EMPLOYED)) 100.)
```

[1]What this really means is that the system doesn't always have to worry about what action got an entity into a particular state (i.e., the (A (B)) form implies that this entity is the result of a particular action on A). Instead, it can simply posit the fact that A is "somehow" in state B. This is often necessary since the causing action may be unknown, very complex, or simply not worth worrying about. (For example, consider (STATE-OF (JOHN) (HANDSOME)).)

NUMBER-OF is somewhat more general in that it can take partially determinate number-substitutes, as in

```
(NUMBER-OF (ORDER (BACKLOGGED)) (MANY)),
(NUMBER-OF (PRODUCT) (DECREASED)),
```

or can remain completely unspecified:

```
(NUMBER-OF (WIDGET (INVENTORIED)) NIL).
```

There are a full range of arithmetic modelling entities for handling NUMBER-OF constructs (SUM, DIFFERENCE, DISCREPANCY [DIFFERENCE if the difference is greater than zero, else zero], RATIO, etc.).

Two modelling entities are used to represent more general functional dependencies between entities (in addition to the totally general "some kind of dependency" represented by DEPENDS-ON). DETERMINED-BY expresses a totally sufficient (i.e., substitutable) functional relationship, e.g.,

```
(DETERMINED-BY (NUMBER-OF (PROFIT) NIL)
    (DIFFERENCE (NUMBER-OF (INCOME) NIL)
                (NUMBER-OF (EXPENDITURE) NIL))),
```

while DETERMINED-FROM simply identifies the relevant factors, leaving open the exact nature of the functional relationship and even the functionality, as in

```
(DETERMINED-FROM ((CUSTOMER) (ORDER))
    (PRICE-EFFECT)
    (DELIVERY-TIME-EFFECT)
    (SELLING-EFFORT-EFFECT)
    (QUALITY-EFFECT)).
```

Here we don't know how (PRICE-EFFECT), (DELIVERY-TIME-EFFECT), etc. are to be used, or even if all of them need to be present for the ((CUSTOMER) (ORDER)) to be determined. The DETERMINED-FROM merely groups them as "all of the known relevant factors". Remember that DEPENDS-ON simply specifies one "known relevant factor" (of which there can be an undetermined number more).

The final structural modelling entities are FLOW, INFLUX, and OUTFLUX. FLOW is the grouping function for flows; that is, the flow

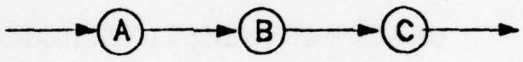


Figure 19. A little FLOW

is represented as (FLOW A B C). This indicates that A B C may be considered as a single flow for purposes of consistency checking, etc., *as we'll see in a second.* INFLUX and OUTFLUX are used to distinguish the input and output flows for any accumulated entity. An accumulated entity is something (like "workers" in figure 17) whose amount at any time is determined by the difference between the amount of that entity which has flowed in and the amount that has flowed out since observation of the system began (plus anything that was there before observation started). The assumption is that the flow in and the flow out are at least somewhat independent so that the accumulated entity is an interesting quantity. In fact, accumulated entities are often there to decouple parts of a flow. Since the inflow and the outflow are decoupled and are usually separately controlled, and since the condition of "outflow when there is no accumulated entity left" is usually an error condition of some kind, the two flows are represented by separate modelling entiities. Thus, figure 17 would be represented as

```
(DETERMINED-BY (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
        (INFLUX (ACT-ON (PEOPLE)
                        (HIRE)))
        (OUTFLUX (ACT-ON (PEOPLE (EMPLOYED))
                         (FIRE)))).
```

INFLUX and OUTFLUX are in all other ways identical to FLOW; in fact, as we will see, it is sometimes useful for the program to consider an INFLUX and OUTFLUX to be a single FLOW for the purposes of cause-effect reasoning.

So far we have considered actions and entities, which are to be associated with pieces of the input description, and modelling entities, which are to be used to interconnect other entities. There is one final category, really a kind of modelling entity, but used in a different way. Modelling entities in this category are to be *associated with* other entities. That is, they are sort of "meta-entities". Examples of these things, which I call **generics**, in the current Alfred system are (DELAY), (CHANGE), (CAUSE) and SMOOTH. (DELAY) indicates that some other entity or group of entities is to be viewed as a delay for the purposes of that part of the model. Thus, the (HIRE) action could be viewed as a (DELAY) if it satisfied the requirements of that particular (DELAY) entity. Because the same model construct may contain several necessarily non-identical (DELAY)'s, the entities (DELAY1), (DELAY2),...,(DELAY9) are recognized as separate "instances" of (DELAY). SMOOTH is the time-averaging function. It is to be associated with the structure of entities which produce the required "smoothing" effect in the model (we will see examples of its use later). It is important not to think of the use of (DELAY) and SMOOTH as the syntactic substitution of some entities for the "meta-entity". It is much more a matter of these "meta-entities" *modelling* other entities much as regular entities model pieces of the problem description. (CHANGE) and (CAUSE) act as "place holders" for an as yet undetermined action which is known to change something in a particular flow or cause a particular effect. As mentioned briefly in Chapter III, generics play a special role in focussing, where their flexibility can be used to change the form of an abstract structure without altering the way it is used by other entities. That is, the fact that an entity is modelled as a SMOOTH means that the other parts of the model that deal with it only care that it is some kind of time-averaging entity--they don't care how the time-averaging is done. The form of the SMOOTH can therefore often be chosen strictly on the basis of problem-dependent considerations. We will see the use of generics fore focussing again in Chapter VI.

The choice and form of these constructs reflect implicit knowledge about the model, as discussed above. They are sufficient (and convenient) for representing all of the kinds of models Alfred is intended to deal with. And, as we will see in Chapter IX, they are actually applicable to a wider domain. But so far we have considered only the relatively small amount of model knowledge that can be contained implicitly in the representation; now it is time to show how the explicit knowledge is attached.

1.2 Knowledge distribution in the abstract model

As I said at the beginning of this chapter, one of the purposes of the abstract model structure is to provide convenient places to put explicit knowledge about the model. In Alfred, this explicit modelling knowledge, i.e., knowledge about how a piece of the model relates to other pieces of the model, can be attached to any entity, action, or modelling entity. It is not to be confused with the abstracters, i.e., knowledge about how a piece of the model relates to pieces of the problem description, which are discussed in section 3 of this chapter.

Explicit modelling knowledge can take several forms. For regular entities and actions, it consists of defining characteristics and, in some cases, possible variations of form. Thus, for the (PEOPLE) entity, attached modelling knowledge would say first, that it was an entity; then, that it could be in the states (EMPLOYED), (LAID-OFF), (WORKING), etc.; and finally, that it is not necessary to model the action which has this entity as its result. The modelling knowledge for the (HIRE) action would begin similarly by declaring (HIRE) to be an action, and then go on to say that it operates on (PEOPLE), putting it in the (EMPLOYED) state. After giving this basic information it goes on to include additional information about models of the kinds of policies that could govern that action. These are in the form of **elaborations**, alternate descriptions at various levels of detail which can be substituted for the entity or action they represent if the need arises. That is, instead of simply positing (HIRE) at some point in the modelling effort, the program can suggest the policy elaboration we saw earlier:

```
(SMOOTH (DISCREPANCY
   ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
        MODIFICATION (DESIRED))
   (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))).
```

That is, hiring is determined by the time-average of the difference between the number of people needed by the firm and the number that it already has (if that number is greater than 0). The way in which elaborations and the other parts of the modelling knowledge attached to entities and actions are used depends on the needs of the modelling effort, as dicussed in Chapter III.

In addition, entities and actions can carry any **characteristics** the modeller wishes to bestow on them. Furthermore, these characteristics can be modified by (NECESSARY) and (POSSIBLE). Thus, the abstract model entity for trainees looks like, in part,

```
(PEOPLE (EMPLOYED (TRAINING)))
   (CHARACTERISTICS
     ((INEXPERIENCED)
          MODIFICATION (NECESSARY))
     ((NUMBER-OF (SENIORITY) (FEW))
          MODIFICATION (POSSIBLE))).
```

("Trainees have low seniority, usually, but they're always inexperienced".) Another kind of characteristic found (rarely) in Alfred's abstract model is cause-effect information like

```
(STRIKE)
    (CHARACTERISTICS
        (CAUSE ((ACT-ON NIL NIL) AGENT (UNION))),
```

i.e., strikes are caused by something the union does. These are expert tricks which can provide shortcuts for the modelling mechanism (often saving a question to the user).

The explicit modelling knowledge that comes with the modelling entities is of a slightly different character (except for the usual "type" information, i.e., the thing that says "this is a modelling entity"). It gives information on how the entity is to be *used*, not only for modelling, but also for reasoning about the model. For example, attached to the DETERMINED-FROM modelling entity is a default use pattern which says (to the modelling process) that for constructs like

```
(DETERMINED-FROM A
                 B
                 C
                 D),
```

don't expect to find an association for A. Instead, associate B, C, and D, and then expect the modelling process to derive A using the associations of B,C, and D. We have already seen that this default way of using DETERMINED-FROM can readily be overridden in specific circumstances. But, it does provide the modelling mechanism with a reasonable strategy if no more specific information is available.

This was an example of how to use modelling knowledge for modelling. In order to see an example of modelling knowledge used for reasoning about the model, let us look at the FLOW construct. I said earlier that FLOW is used to group actions and entities into a single flow. But it also has attached knowledge about what it *means* for actions and entities to be in a single flow (referring to figure 19):

(1) A change in A may affect B or C.

(2) A change in B or C may affect A (though not as directly).

(3) Anything flowing through A must flow through B and C (though perhaps in a different form) unless specifically accounted for elsewhere (e.g., "still sitting in inventory").

(4) Only the state of an entity can change as it moves through the FLOW; at any point in the FLOW, the basic entity must be the same.

(5) The FLOW can be considered as an "action" with the input characteristics of A and the OUTPUT characteristics of C.

Knowledge like this is also attached to DELAY, SMOOTH, INFLUX, and OUTFLUX.

Another example of this kind of "reasoning about the model" knowledge is the feedback analysis information that is attached to DETERMINED-BY. This consists only of a few simple heuristics: how to go around a feedback loop, how to see if it is positive or negative feedback, how to find the order of the loop. For example, the order of the loop is determined by the number of accumulation entities, the "storage" parts of the flow that correspond to the state variables of differential or difference equations that describe the

system (see [Forrester]). The program uses this kind of information to see, for example, whether a given DETERMINED-BY structure could give rise to oscillations (must describe a loop of at least the second order (only second order or greater equations can describe curves with oscillatory behavior)), or could cause growth (must contain positive feedback). That is, the above heuristics are important because they allow the program to deduce important characteristics of the system's behavior without having to simulate or solve differential equations. This is really the only kind of feedback analysis that I have found necessary for doing the sorts of reasoning that Alfred must do to solve problems. Simulation is used only as a check--which has the tremendous advantage that certain characteristics of a particular variable can be picked out for observation: they will either confirm or deny the hypothesis under consideration. For example, if a suggested policy is supposed to reduce fluctuation in some variable in the system, tha variable can be checked during simulation to compare its frequency before and after the suggested policy change is implemented. If the frequency is the same or higher than before, the policy suggestion isn't working. Of course, there is a great deal more to be said about feedback systems in terms of stability determination, control strategy, etc. (although most of it is very hard to use for non-linear systems). I believe that this kind of knowledge could be added fairly readily to the simple heuristics of DETERMINED-BY, but it has not been necessary.

Similarly, there is "how to use me" knowledge attached to other modelling entities like (CHANGE) (if (ACT-ON y x), then x may (CHANGE) y, etc.), (DELAY) (characteristics of first and third order delays, and how to use them), SMOOTH (how to recognize a particular modelled structure as a SMOOTH, how to choose the time averaging constant, etc. (this is really a kind of (DELAY), in terms of semantics)). This information is used in several ways by the theme, trend, edge mechanism, as we will see in Chapter VII.

This about wraps it up for the kinds of explicit knowledge that can be associated with pieces of the abstract model. I will now go on to describe the particular selection of actions and entities that make up Alfred's abstract model, the model that will be used for examples throughout this thesis.

1.3 Alfred's abstract model

The abstract model implemented in the current Alfred program contains the *workforce need theory* sketched in I-1. As mentioned there, workforce fluctuation, the need to hire and layoff employees "too often", is a frequent complaint of managers. The managers usually consider the sole cause of this to be unavoidable fluctuation in the workload, which forces corresponding changes in the personnel level of the firm in order to maintain the appropriate balance between labor costs and firm productivity. However, for the consultant, this explanation covers only the *basis* of the problem. Expert analysis often reveals that the "unavoidable" component of the workforce fluctuation is fairly small; most of the hiring and firing is needless--due to policies which heighten the unavoidable workload fluctuation rather than smoothing it (or leaving it the same) as they should. Alfred must therefore look at the firm and decide exactly what is causing the workforce fluctuations. The program can "diagnose" several different problems:

(1) The workforce fluctuation *is* completely explicable in terms of workload fluctuation; no "amplification" is detected. This is considered "normal", and the program has nothing to recommend to fix it. (Note that it could still be undesirable, in that hiring and firing costs could be greater than the cost of maintaining excess personnel, and fixable, in that advertising or bringing in new product lines could actually smooth the demand--but this is beyond the program's realm of expertise).

(2) Failure to take into account delays inherent in the system causes amplification of the normal fluctuations. This is a common problem in many kinds of systems: the manifestation of it here is that the fact that hiring and firing decisions take effect "out of synch" with the demand fluctuations is not recognized by the management, who find themselves continually "correcting" the employment level. Of course, the corrections never quite catch up with the real fluctuations, and often end up exacerbating rather than ameliorating them:

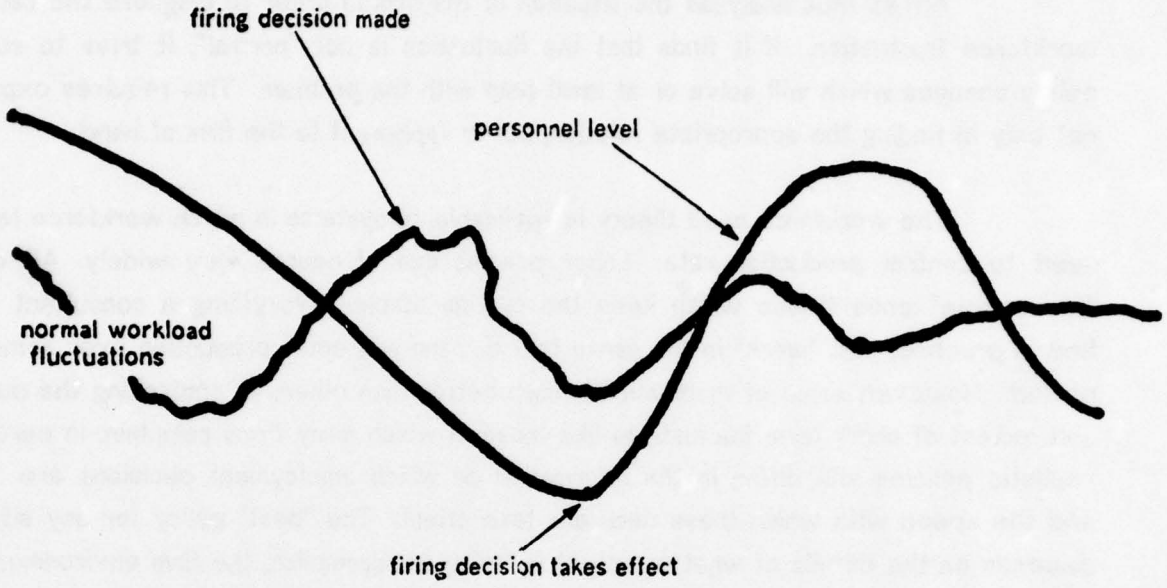


Figure 20. Decision effect lag exacerbates fluctuations

(3) Finally, Alfred's speciality: the "overshoot" problem described in I-1. The management of the firm bases its hiring and firing decisions on the number of workers needed to handle the current workload, and considers labor level to be the only factor which reduces or increases production needs. The firm thus ignores factors like decreased sales due to delivery delay on backlogged orders (customers perceive a significant delay in receiving services and transfer their business elsewhere) or increased sales due to high inventory (which naturally increases it). The result is that personnel levels constantly overshoot goals, because the management's hiring and firing decisions overcompensate for workload fluctuations. Furthermore, this often turns into an out of synch effect (a la (2))--even if delays are considered properly--because part of the need for the employment adjustment has disappeared (by natural causes) just about the time adjustment takes effect. Again, attempts to "re-correct" by more hiring and firing only make matters worse.

Alfred thus analyzes the situation of the firm in order to diagnose the cause of workforce fluctuation. If it finds that the fluctuation is not "normal", it tries to suggest policy changes which will solve or at least help with the problem. This requires expertise not only in finding the appropriate solution, but in *applying* it to the firm at hand.

The workforce need theory is applicable to systems in which workforce level is used to control production rate. Labor policies can of course vary widely. All of the "reasonable" ones (those which keep the system stable--everything a consultant would find in practice) will "work" in the sense that demand will equal production over some time period. However, some of them will be much better than others at controlling the duration and extent of short term fluctuations like those of which many firms complain. In particular, realistic policies will differ in the information on which employment decisions are based, and the speed with which these decisions take effect. The "best" policy for any situation depends on the details of what exactly it is trying to accomplish, the firm environment etc. Therefore, the suggested solution to a problem depends completely on the exact nature of the case that is presented, and I will defer discussion of the choice and application of solutions until the detailed example of Chapter VIII.

But before Alfred can think about diagnoses and treatments, it must be able to understand, i.e., model, the symptom and the firm itself. It turns out that a variety of symptoms can signal workforce fluctuation problems. The program's methodology for discovering and understanding these symptoms is the topic of the next section. As far as modelling the firm goes, for the workforce need theory this translates into modelling (at some level of detail) the workforce, customer ordering, and production sectors. It would be much too long and tedious to include a full description of what the program knows about each of these sectors. However, I do wish to include a few highlights in order to give an idea of the kind of modelling the program must do in order to apply its theory. Accordingly, a selection of the relevant actions and entities is given below.

WORKFORCE SECTOR

```
(DETERMINED-BY
  (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
    (INFLUX (DELAY1 (ACT-ON (PEOPLE)
```

```
                                    ((FIRM) (HIRE)))))
(OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                         ((FIRM) (LAYOFF))))))
```

```
((FIRM) (HIRE))
      (SMOOTH (DISCREPANCY
        ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED))
        (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)))
```

--this is a typical model of a policy action: a continuous attempt to eliminate a defined discrepancy between the desired level of an effect and its actual value; remember that DISCREPANCY has value 0 if its argument is less than 0

```
((FIRM) (LAYOFF))
      (SMOOTH (DISCREPANCY
        (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
        ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED))))
(DETERMINED-BY
      ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED)
        (SUM
            (RATIO (SMOOTH ((CUSTOMER) (ORDER)))
                   (PRODUCTIVITY))
            ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
               MODIFICATION (DESIRED EXTRA)))))
```

--that is, the number of employees desired is equal to the number needed to handle the current order rate (which is defined to be the current order rate over the productivity of the individual employee) plus "extra" personnel, as defined below; note that there may by a negative number of extra personnel needed, e.g., if the firm has an inventory from which to fill orders

```
((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
```

```
        MODIFICATION (DESIRED EXTRA))
    ELABORATIONS
        ((TIMES (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
                (ATTRITION))
        (RATIO (SMOOTH (DIFFERENCE
                    (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                    ((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                        MODIFICATION (DESIRED))))
               (PRODUCTIVITY))
        (MINUS (RATIO (SMOOTH (DIFFERENCE
                        ((NUMBER-OF (PRODUCT (INVENTORIED)) NIL)
                            MODIFICATION (DESIRED))
                        (NUMBER-OF (PRODUCT (INVENTORIED)) NIL)))
                      (PRODUCTIVITY)))
        (RATIO (SMOOTH (DIFFERENCE
                    ((NUMBER-OF (PRODUCT) NIL)
                        MODIFICATION (EXPECTED SEASONAL))
                    (NUMBER-OF (PRODUCT) NIL)))
               (PRODUCTIVITY))
        (RATIO (SMOOTH
                   (NUMBER-OF (ORDER (BACKLOGGED)) NIL))
               (PRODUCTIVITY)))


(PRODUCTIVITY)
  ELABORATIONS
      ((CONSTANT)
      (DECREASING-FUNCTION
            (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))
      (INCREASING-FUNCTION
            (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))
      (HUMP-FUNCTION
            (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)))
```

--individual productivity may be independent of the number of workers, or may depend on it in any of several ways

```
((FIRM) (TRAIN))
   ELABORATIONS ((DELAY (PEOPLE (EMPLOYED))))
```

--again, I list only the one

CUSTOMER ORDER SECTOR

```
(DETERMINED-FROM ((CUSTOMER) (ORDER))
      (PRICE-EFFECT)
      (DELIVERY-TIME-EFFECT)
      (SELLING-EFFORT-EFFECT)
      (QUALITY-EFFECT))
   ELABORATIONS ((DECREASING-FUNCTION
                  (RATIO (ORDER (BACKLOGGED))
                         (PRODUCTION))))
```

--customer ordering is in general a complex activity based on a combination of the four factors shown; it is therefore modelled as a DETERMINED-FROM which means (just as it did for SYMPTOM determination from SUSPECT-ACTIONS and SUSPECT-EFFECTS) that the other arguments should all be investigated first to determine which need to be considered as relevant; for simplicity I show only the ELABORATION which handles a single DELIVERY-TIME influence

PRODUCTION SECTOR

```
(DETERMINED-BY (ORDER (BACKLOGGED))
      (INFLUX (ACT-ON (ORDER)
                      ((CUSTOMER) (ORDER))))
      (OUTFLUX (DELAY1 (ACT-ON (ORDER (BACKLOGGED))
                               (PRODUCTION)))))


(DETERMINED-BY (PRODUCT (INVENTORIED))
      (INFLUX (DELAY1 (ACT-ON (RAW-MATERIAL)
```

```
                                    (PRODUCTION))))
(OUTFLUX (DELAY2 (ACT-ON (PRODUCT (INVENTORIED))
                         (SELL)))))


(PRODUCTION)
   ELABORATION (((FIRM) (ORDER-FILLING)))


(DETERMINED-BY (ORDER-FILLING)
      (TIMES
            (PRODUCTIVITY)
            (NUMBER-OF
              (PEOPLE (EMPLOYED (WORKING)))
              NIL)))
```

--this particular concept of ORDER-FILLING is the standard default for situations in which the details of production are not given (and are presumably unimportant): production is simply taken to be a dummy "order-filling" activity by which customer orders just go away; specifically, no (PRODUCT) is produced.

Section 2 Symptom-finding knowledge

Although the symptom-finding knowledge of the program is expressed in terms of abstracters (see the next section), I think that it deserves special mention as a separable part of Alfred's knowledge base. As discussed in the previous chapter, the program is rarely told what symptoms it should work on; it has to figure it out. That is, it has to look at the problem description, see what features might be symptoms of a problem, and decide which symptoms to work on. The program uses two modes of analysis to answer these questions.

The first is basically a linguistic technique: the program looks for features which occur *in spite of* some stated desire of the client. Thus, if the problem description says that "our order backlog is increasing in spite of our policy to keep it constant" (or some (close)

linguistic equivalent), the program assumes that "increasing backlog" is a (SUSPECT-EFFECT), i.e., a possible symptom. (SUSPECT-EFFECT)'s and (SUSPECT-ACTION)'s found on the basis of this "in spite of" analysis are usually the best candidates for symptoms.

The other analysis technique assumes that the client will not always explicitly say that he doesn't want a particular thing to be happening. He will assume that the program knows it's a "bad thing". Thus, Alfred has a notion of a few "bad things" such as

```
(NUMBER-OF (PRODUCT (RETURNED))
           (MANY))

(STATE-OF (NUMBER-OF (PROFIT) NIL)
          (DECREASING))

(STATE-OF (NUMBER-OF (WORKFORCE) NIL)
          (FLUCTUATING))

(DEPENDS-ON (STATE-OF (FIRM)
                      (NIL
                       MODIFICATION (BAD)))
            ((ACT-ON NIL
                     NIL)
             AGENT (UNION)))
```

--i.e., "union troubles"

```
(STATE-OF (NIL
           OF (FIRM))
          (UNMANAGEABLE))
```

--i.e., some aspect of the firm is being controlled by external forces, is in effect uncontrollable by the firm...a dangerous situation

The use of these bad things is not as straightforward as the use of "in spite of" analysis. First of all, bad things are weaker evidence for a symptom: the program is simply assuming that effects like this are undesired[1]. Second, bad things are made up of entities of the abstract model. In order to see whether the client has actually described a bad thing in his input, it is necessary to go through a modelling effort, just as for any other aspect of the abstract model. (E.g., the client might describe what is here given as a STATE-OF as some actions he has been forced to take, etc.) Finally, it is often the case that a bad thing is a feature not only of the client's firm, but of the whole industry as well. If this is true, Alfred assumes that though the thing is still bad, it is probably *not* the problem the client has come to see it about. The assumption is that a problem is only a problem relative to competitors. Therefore, before calling a bad thing a (SUSPECT-ACTION) or (SUSPECT-EFFECT), the program has to make sure that a similar bad thing has not been given as a characteristic of the industry. This of course requires another modelling effort, including some notion of what "similar" means (see VII-3.2). Thus, the use of "bad things" is less reliable and more complicated than the linguistic "in spite of" analysis.

Once the program has found all of the (SUSPECT-ACTION)'s and (SUSPECT-EFFECT)'s by either of these two methods, it proceeds to decide on a (SYMPTOM) (or, actually, a (SYMPTOM (UNDER-CONSIDERATION))). It siezes upon one of the suspects (looking for those found by "in spite of" analysis first) and checks the rest of the list to see if there are any "supports" for this symptom, i.e., if any are part of the same symptom. This is not an intensive effort, but rather a quick check to see if some of the other suspect actions and effects might be dealing with the same entities of the abstract model. (It's useless to go in much deeper at this early stage of the problem-solving effort; the program can't really decide whether two features are actually part of the same problem until it knows a great deal about the problem (if then).) The program chooses its best supported "in spite of" (or, if none, "bad thing") symptom as its (SYMPTOM (UNDER-CONSIDERATION)), and proceeds to try and solve the problem it presents. The other symptoms are saved for later examination if the program stalls or is redirected by the user. (I have not yet run across a case with more than two reasonable symptoms.)

[1]That is, Alfred assumes that in the environment it is working in, these effects are undesirable. If, for example, a parent company were using the firm described in a case as a tax loss, "decreasing profits" could be a desired effect, and Alfred's symptom-finding mechanism could end up barking up the wrong tree.

Section 3 The abstracters

We will see in the next chapter that one of the fundamental goals of the implementation philosophy behind Alfred is to eliminate the control structure role of the expert association tricks I call abstracters, and to provide a problem-solving organization in which to use them in a straightforward manner. The reason for this is that the abstracters in a model to be used to be used for reformulation may be uncertain or unorganized. Thus, though absolutely essential to the problem-solving effort, they should not be used to run the show, as they do in frame-type methodologies (that is, as we have seen , the same kind of knowledge that goes into abstracters in Alfred becomes the control links in frame systems). Therefore, abstracters, as used by Alfred, are specialized database access routines which are called by other, more global, procedures.

The format of abstracters is determined by ease of writing and ease of explanation. They are composed of calls to function-writing functions which construct a particularized database access function that can do the needed search or lookup in a straightforward and efficient manner. The particularized access functions are written without regard to readability. However, the calls to function-writers which actually make up the abstracters are designed to be free of the vagaries of the database structure and, more important, to be readily understandable and explainable.

For example,

```
(COLLECT '(ALL (UNDER '((FIRM) (POLICY))) (OF-TYPE ACTION)
               (OR (UNMANAGEABLE) (DEPEND-ON (UNMANAGEABLE)))))
```

expands into an efficient access routine which collects all policies of the firm (i.e., all constructs of type "ACTION" which are under the "((FIRM) (POLICY))" property) which have (UNMANAGEABLE) as a characteristic or modification, or depend on an unmanageable action or entity. Similarly,

```
(LOOKUP '((FIRM) (QUALITY-CONTROL))
   '(1 (FOR (COST-OF (QUALITY-OF (PRODUCT))))
     (PROPERTIES MANAGER-DESCRIPTION
                 CHARACTERISTICS
                 EFFICIENCY)
     (IF ((PRODUCT) OF (FIRM)))))
```

means that the program should look up under the ((FIRM) (QUALITY-CONTROL)) entity in order to find out something about (COST-OF (QUALITY-OF (PRODUCT))). Specifically, one ("1") of the things that the manager says about the efficiency of the firm's quality control process may be applicable to the cost of product quality, provided that the manager is talking about quality control as it relates to a product of the firm. In practice, this means that the first thing that the abstracter finds under the EFFICIENCY-OF property of the CHARACTERISTICS property of the MANAGER-DESCRIPTION property of ((FIRM) (QUALITY-CONTROL)) whose object is ((PRODUCT) OF (FIRM)) (or ((PRODUCT (any state)) OF (FIRM))) will be placed under (COST-OF (QUALITY-OF (PRODUCT)) for further examination by whoever called the abstracter in the first place. (This is about as complex an abstracter as you're liable to run across.)

Abstracters are the parts of the program which actually make the associations between abstract model pieces and pieces of the problem description. Thus, perhaps the first kind of abstracter that comes to mind is the general data-gathering kind of function seen in many programs. For example,

```
(COLLECT '(ALL (UNDER IN-SPITE-OF) (OF-TYPE ACTION)
               (IF (AGENT (FIRM)))),
```

which finds all of the firm's actions that occur in spite of something. But there are really very few such abstracters: only those attached to entities like (SYMPTOM) and (THEORY) about which little is known intrinsically. These entities are really just simple place-holders for the objects of interest, and their abstracters are necessarily broad-ranging methods for finding the real objects without knowing anything very specific about them. As I said earlier, though, this isn't the way the program is set up to work in general. It relies much

more on very specific bits of association knowledge (and, implicitly, on the mechanism which knows how to use them).

Therefore, the majority of Alfred's abstracters are action- and entity-specific "hints" or tricks for making associations. These are on the order of (under (PRODUCT)) "if you're looking for information about product quality (and there is none), look for the customer's image of that quality", or

```
((FIRM) (PRODUCT))
   (CHARACTERISTIC
      (QUALITY
         ABSTRACTERS (LOOKUP '((QUALITY-PERCEPTION)
                                 OF (CUSTOMER))
                             '((FOR (QUALITY-OF (PRODUCT)))
                               ALL[1])))).
```

Another kind of entity- and action-specific abstracter is the "similarity information" used by the trend (see VII-3.2) to find out whether one structure is substitutable for another. An example is "firing is the same thing as layoff unless 'people that have been laid off' is a distinction made by the firm":

```
((FIRM) (FIRE))
   (ABSTRACTERS (LOOKUP '((FIRM) (LAYOFF))
                        '((FOR ((FIRM) (FIRE)))
                          (IF (NULL (LOOKUP '((FIRM)
                                    (PEOPLE (LAID-OFF)))
                                    '(PROPERTIES
                                      MANAGER-DESCRIPTION)))))),
```

(i.e., lookup information under ((FIRM) (LAYOFF)) and copy it to ((FIRM) (FIRE)) unless the manager has defined a (PEOPLE (LAID-OFF)) entity).

[1] If only some aspects of QUALITY-PERCEPTION can be used for (QUALITY-OF (PRODUCT)), which is undoubtedly the case in any complex problem, "ALL" would be replaced by the appropriate filtering predicates.

There is a considerable amount of this kind of unstructured information which the program must be able to deal with. As I will show in the next chapter, the advantage of the programming methodology used in Alfred is that this information can simply be put under the entity or action it belongs with and "left there".

In addition, abstracters have access to, and often use, the "necessary" and "possible" characteristics kept under actions and entities. They form another aid to the association process, since they provide evidence (especially negative evidence) for whether the object described in the problem description is really what the abstracter thinks it is. Suppose, for example, that the program were trying to associate the (QUALITY-CONTROL) action of the abstract model with something in the problem description. Now suppose that one of the statements about product quality in the problem description were

```
(DEPENDS-ON (NUMBER-OF (COST-OF (PRODUCTION)) (DECREASED))
            (STATE-OF (QUALITY-OF (WIDGET) (INCREASED)))).
```

That is, if widget quality is high, production costs are decreased. The program would immediately determine that this statement did *not* refer to the (QUALITY-CONTROL) action of the firm, since (QUALITY-CONTROL) has as a necessary characteristic that higher quality means higher production costs:

```
((FIRM) (QUALITY-CONTROL))
    ((CHARACTERISTIC
        (DEPENDS-ON (NUMBER-OF
                        (COST-OF (PRODUCTION))
                        (INCREASED))
                    (STATE-OF
                     (QUALITY-OF ((PRODUCT) OF (FIRM))
                     (INCREASED)))))
      MODIFICATION (NECESSARY)).
```

(The implication is that "widgets" are products which are bought by the firm and used in production. The higher *their* quality, the lower the production costs of the firm.) Similarly, the necessary characteristics can be used to *suggest* an association, as in the abstracter attached to (SUSPECT-EFFECT) which looks for anything that is (UNMANAGEABLE).

I don't want to give the impression here that abstracters devote a great deal of their energy to this recognition-style association by characteristics. Actually, quite the reverse is the case. Characteristics are mainly used for making associations which are not derivable by other means; the connection between cost and quality control shown above is a good example. Most association work is done by the more straightforward specific entity-to-input abstracters like those given earlier, which are used by the reasoning processes of the trend (see VII-3) to find the right match-ups. The entity-characteristic-to-input-characteristic kind of abstracter is reserved for those cases in which it is the only known connection device (as in the (UNMANAGEABLE) thing above), or in which it provides a quick (almost always negative) check as in the quality control example. This is not to say that the characteristic-style abstracters *cannot* be used everywhere, it's just that they're usually not necessary. If there is some reason for relying on lots of characteristic-based expert tricks, the modeller can just put them in with the rest of the abstracters.

Finally, I must point out that abstracters may be attached to any piece of the abstract model, not just simple actions and entities. If there is an expert trick which relates to a larger structure within the model, it too can be expressed as an abstracter and attached to the structure for use by the modelling effort. For example, the employment flow model actually used by Alfred contains (as we saw in 1.3), in addition to the constructs shown in figure 17, the possibility of a (DELAY) between "hire" and "workers" (in the parlance of the figure). There is an abstracter attached to this flow which says "if (DELAY) exists it may be the (TRAIN) action of the firm" (there are other alternatives as well). Note that this little relation is certainly not a property of (DELAY) and wouldn't do much good attached to (TRAIN). It is a property of that particular flow, and that is where it must be attached in the abstract model. There are a number of these very specialized abstracters scattered throughout the modelled system. Clearly, they are not useful for anything except the particular piece of structure they deal with. But, if that piece of structure happens to come under consideration, they can make the modelling process immensely more efficient. I think that it is reasonable to expect experts to have some of these very particular bits of knowledge, and that every expert program methodology should have some facility for dealing with them.

Section 4 The modelled system

There is one final part of the Alfred system which must be characterized as a piece of the knowledge base. This is the modelled system, the tailoring of the abstract model to the problem situation at hand. The representation of the modelled system does not differ significantly from that of the abstract model. However, its uses are fundamentally different. The most important use of the modelled system is as a dynamic repository of knowledge about the problem-solving effort, the source of the theme, trend, edge information. I will reserve discussion of this aspect of the modelled system until Chapter VII. It doesn't really belong in this chapter anyway, since in *this* use, the modelled system is not strictly a knowledge base under our definition: it does not limit the range of applicability of the program. Rather, it is a database in which the theme-, trend-, and edge-setting mechanism (itself independent of the domain of the program) looks for information to guide the program control structure. We will explore this distinction in Chapter IX.

However, after the modelling effort is complete, the modelled system does become a knowledge base for the program. It represents the domain in which the firm's problem and its possible solutions are analyzed. It is also the environment in which the proposed solution is applied and tested.

After the symptom-finding effort has been concluded, the program has a good idea of the evidence for the problem. But this is not at all the same as knowing the cause of the problem. In some problem domains (e.g., some parts of medicine), knowing the symptoms and the situation is only the beginning of an intensive diagnostic effort. Fortunately, the strong structuring and thorough knowledge of the abstract model make the symptom-*cum*-model a strong indicator of possible causes of the evidenced problem. Nonetheless, some analysis is usually necessary to choose between similar possibilities for the cause (e.g., two or three choices in Alfred). This analysis must be done in the modelled-system--it represents the program's total knowledge of the constraints and features of the problem situation, as seen from the point of view of the abstract model. Since it is an abstract-modelled structure, it yields itself to the powerful analysis techniques (tracing cause-effect chains using knowledge about FLOW's, following feedback

loops and doing feedback analysis, simulation and observation (see below), etc.) that come with the abstract model, while constraining their use to the features of the particular problem at hand.

Once a cause has been ascertained, a solution to the problem can be selected. Again, some analysis may be necessary to see if this solution can really be applied to the situation at hand. In other words, are there any features of the problem situation which prohibit the use of the proposed solution or reduce its effectiveness? Since the modelled system contains all of the program's knowledge about the constraints of the problem, it acts as the knowledge base for this analysis.

Furthermore, it turns out that the conceptual structure and representation of the abstract model lead to a modelled system which is a workable simulation model of the firm (assuming that "initial values" of the appropriate variables are solicited or supplied). The modelled system is therefore a simulation knowledge base in which proposed solutions can be tested. The solution is applied to the modelled system, the system is simulated, and the results are observed (i.e., certain key variables are watched). If the undesired features are ameliorated or the the desired ones enhanced, the solution is presumably a good one. The program (or the user) can thus gauge the effectiveness of the solution by observing crucial variables in the simulation before and after the solution is applied.

The modelled system is thus the working knowledge base of the program after the modelling effort has been completed. It is actually the mutual knowledge base of the consultant and the client, representing their mutual understanding of the problem. Finally, it is demonstrable evidence of the effects of the solution, and an appropriate simulation model of the firm.

Chapter VI
Connection

The knowledge base of virtually any expert program represents some kind of condensation of the input domain, because the range of possible input specifications is almost always greater than the range of knowledge base specifications. Therefore, a certain amount of effort is inevitably necessary to form an association between the input and the relevant piece of the program's knowledge base--or to show that the piece of input is "irrelevant", i.e., has no association with any piece of the knowledge base. I will call the process of making associations between the input and the expert knowledge base "connection". (This is in order to stay away from loaded words like "recognition", and the word "matching", which many people, including myself, reserve for the necessary subprocess of connection which is concerned with determining whether a particular item constitutes a valid instantiation of a *syntactically* more general pattern (for a detailed discussion of the matching process itself, see [McDermott and Sussman] and [McDermott]).)

Now certainly programs, even expert systems, differ widely in their emphasis on connection and the amount of their total problem-solving effort they are willing to devote to it. In this section, I wish to focus attention on those systems which must put considerable effort into connection before they can apply their knowledge. This means any expert system which has a large real-world domain, or gives the user the opportunity for a realistic variety of expression, or whose knowledge base is in a form which is significantly different from the form of the input. I think that this includes almost all expert systems of current and future interest. There is of course no absolute distinction between programs that must put considerable effort into connection and those that need not. In fact, it is sometimes not clear where the connection effort of an expert system *is* (for example, the straight-shooters subsume some of their connection task in the digestion process which occurs before the program is ever written). Nonetheless, I think that there is a consistent qualitative distinction between programs which apply their knowledge to a wide enough variety of inputs to require some sort of problem-solving in their connection methodology and those which restrict input sufficiently to enable them to do connection in a straightforward manner.

This connection business takes on especial importance here, because, as mentioned in Chapter I, the necessity of an intelligent connection process is one of the major characteristics of, in fact, is part of the definition of, the reformulation method of problem-solving. I will therefore use this chapter to examine the major design decisons that went into Alfred's connection methodology.

Any connection process is involved with the linking of a piece of knowledge base to a piece of input.

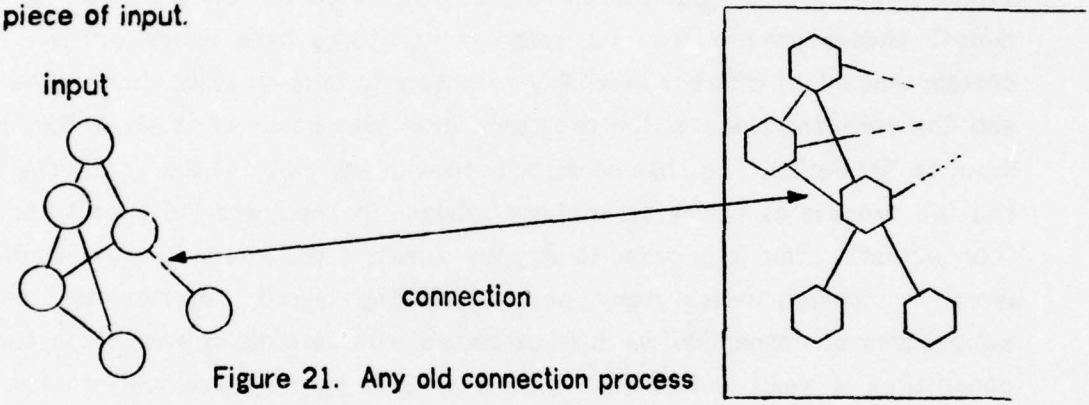


Figure 21. Any old connection process

The major design issue is *where* to concentrate the problem-solving effort needed to make this link:

(1) It can be put into *input-driven* activities which successively refine the piece of input until it is in a form that can be matched to the appropriate chunk of knowledge base.

(2) It can be put into the individual chunks of the knowledge base, some of which are directly *triggered* by pieces of the input (unprocessed), and others of which are triggered by other chunks.

(3) It can be put into a chunk-"massaging" procedure which, given a particular piece of input, processes a "sort of right chunk" (chosen on the basis of non-connection-oriented information (e.g., structure of the model) or by

simple rough-cut heuristics) into something "right" enough to be matched to the input. That is, it *focusses* the chunk of knowledge base onto the piece of input.

Of course, any real system may use a combination of these three approaches, but the expert systems I have looked at always seem to emphasize one or another of the alternatives to a preponderant degree (for example, early vision systems [Winston] and MACSYMA [Moses] use successive refinement; MYCIN [Shortliffe], HACKER [Sussman], and the system proposed in [Rubin] use triggering; Marr's system [Marr and Nishihara] uses focussing). I will therefore start by considering each methodology separately and then worry about how they combine at the end of the chapter.

The first approach, successive refinement of the input, is used in most existing systems. The connection process is comprised of a series of transformations, always input-driven, each of whose goal is, not a particular chunk of the knowledge base, but rather a new representational form of the input. Each transformation makes this representational form closer to the form of the knowledge base, so that in the final phase the refined input is within an easy match of the applicable chunk of knowledge base.

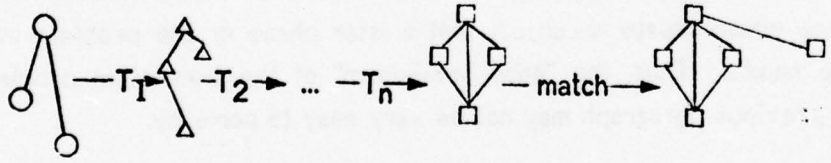


Figure 22. Connection by successive refinement of the input

An important characteristic of this approach is that, the transformations (like the "T_i" above) are based not on specific properties of the particular piece of input or the particular chunk of knowledge base, but on pre-set, general knowledge of "what should be done" to the input at each phase. Thus, in an old style vision system, there would be a

"line-finding" phase in which intensity points are aggregated into lines, followed by a phase in which lines are parsed into regions, and finally by one in which the regions are composed into the objects which are matched against the patterns of problem-solving procedures for doing work in the blocks world. A more familiar example is the usual compiler, which works in input-driven phases to associate the source language input with its appropriate translation into object code.

Connection by successive refinement has the advantage that the individual transformation phases are almost totally decoupled: each phase simply takes its input from the output of the previous phase. The whole connection problem can thus be broken down into a progression of separate subproblems. Furthermore, a good breakdown makes it relatively easy to do error-checking at each phase in order to ensure that the previous phase has produced consistent, usable output, and that the overall connection effort is going well.

This approach relies on two things: the ability to formulate the whole connection process as a series of translations between separate levels of description, and thorough knowledge of what constitutes legal input at each phase. This second consideration is not as straightforward as it looks, because "legal" means not only legal for a particular phase, but also *eventually* legal for the deductive parts of the system. That is, the whole process, though input-driven, must be carefully designed so that no untoward transformations are made at any phase which create an object that a later phase or the problem-solver will not know how to handle. Thus, the "good breakdown" of the translation problem that I mentioned in the previous paragraph may not be very easy to come by.

Remembering the characteristics of expert systems, especially reformulation systems, discussed in Chapter I, we can see that successive refinement of the input, like the other three methods, requires a particular expert system environment in which to work. First of all, it must be fairly clear to the expert a priori what form the client's input is going to take (or the expert must force a structure on the client's input process--doctors seem to do this, see [Miller]). If the system allows appreciable freedom in the description of input problems (as Alfred does), it is much harder to create a succive refinement process which can ensure at any intermediate stage that the input processed so far will eventually turn into something which can be handled by the problem-solver--i.e.,

be legal in the sense discussed above[1]. Furthermore, unless the input is structured in an expected form, it may be difficult for input-driven processes to decide at the initial or some intermediate phase which parts of the input should be ignored, which are important, what's missing, etc. The level of detail and model-tailoring considerations mentioned earlier are therefore particularly difficult for successive refinement type processes to handle in an environment which allows appreciable freedom in the form of the input. Essentially, if the expert knowledge in the program is more of the form of what the expert needs to know rather than of what to do with what it happens to be told, successive refinement of the input is probably going to be inconvenient. Finally, experience has shown that it is quite difficult to resolve complex connection problems into a series of transformations between separate levels of description (see the motivation for *heterarchy* in the introduction to [Winston]). Usually this is because consistent "levels" of description for all of the objects in the domain cannot be defined. A more fluid, interactive transformation mechanism is often necessary in these cases.

It seems clear, then, that in an environment in which relatively little is known about the input, approach (1) is going to be harder to use, simply because it is fundamentally input-driven. In such an environment, in which the program presumably knows most about its own expert model, it seems reasonable to have the connection process be basically model-driven. Alternatives (2) and (3) can both provide model-driven connection processes, but they differ considerably on how the model based connection knowledge should be used. Alternative (2), which ought to be clearly recognizable as the basis of the "frame approach" after the discussion of Chapter II[2], counsels the development of an array of chunks which are triggered by strictly local information in the form of input cues or calls from other chunks. The connection process thus ends up being the "path of triggering" from the piece of input to the final triggered chunk (see figure 23). Alternative (3) suggests the use of fewer, "more abstract" chunks (i.e., each can apply to a wider variety of input pieces than alternative (2) chunks) and the development of a

[1]Note that the digestion phase of the straight-shooters is an attempt to guarantee that legality: the knowledge base of the system is constructed taking the input into account so that connections are guaranteed to be handleable by the system. This is not to say that straight-shooters must use input-driven connection processes, but rather that they must inevitably consider some of the same issues that come up in that approach.

[2]Many proposed frame systems are intended to be basically input-driven. I am not considering these here. An example of a model-driven frame system can be found in [Rubin].

"focussing procedure" which is good at changing a given abstract chunk into something which can be matched with a given piece of input (see figure 24). The procedure has access to both local and non-local information.

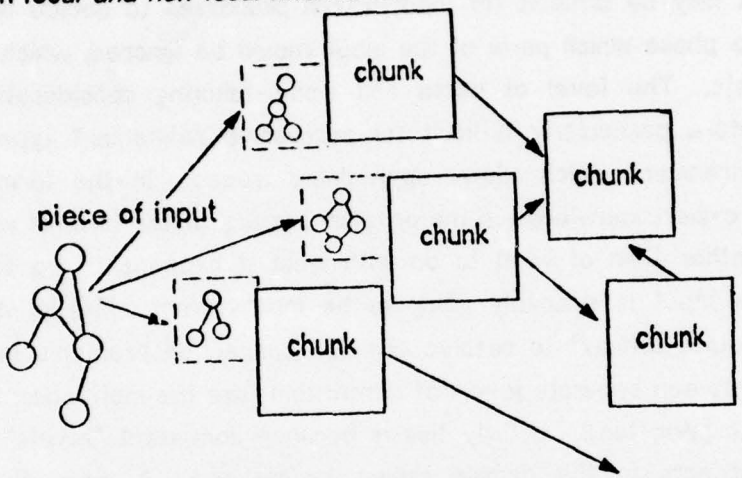


Figure 23. Connection via a triggering mechanism in each chunk

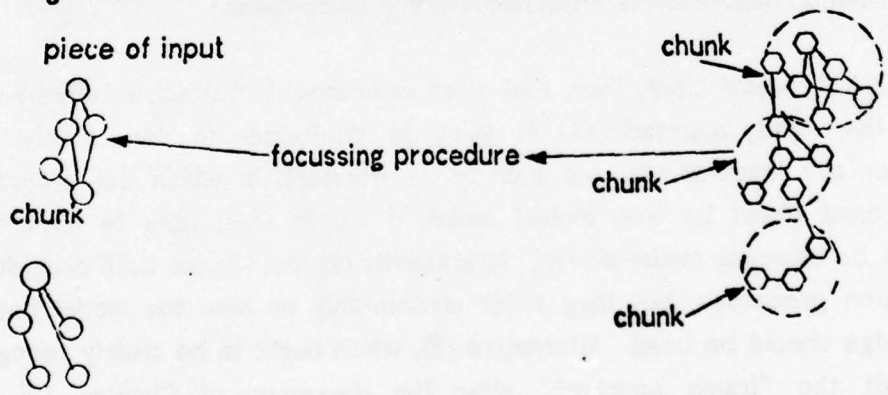


Figure 24. Connection via focussing chunks onto input

Now in Chapter II we examined the way in which basic philosophies of building expert systems bias the structure of the problem-solving process. Here we will look at

things from the other perspective: what do these two structures for the connection process have to offer the builder of expert systems?

The basic advantage of doing connection with the built-in triggering mechanism of the knowledge chunks is that it provides a convenient generate-and-test kind of control structure in which "plausible chunks" are suggested for quick verification against the input. This in turn makes it easy to recover from "failures" (i.e, suggested chunks which turn out to be inapplicable) while retaining the connection knowledge learned so far (i.e., the useable cues that have already been ferreted out of the input). In fact, the usual proposal is that these systems should use the "error"--the cue or set of cues which caused the particular chunk to "fail"--to suggest another possible chunk (see, for example, the proposals of [Kuipers], [Rubin], and [Minsky]). This all comes about because the connection processing for each chunk is contained in that chunk. All of the connection knowledge which bears on the chunk can therefore be manipulated (pushed, popped, stuck into contexts, etc.) as a "chunk"--i.e., as a pretty much independent piece of the knowledge base. A final advantage which usually accrues to the chunk triggering method is that there is a very clear notion of successful connection, i.e., the first plausible chunk which is totally verified with respect to the input.

On the other hand, the triggering approach has the disadvantage I mentioned earlier in connection with frames: it is quite difficult to communicate the effects of the application of one frame throughout the system in order to constrain or aid the overall problem-solving effort. That is, it is very hard to get the global view of the chunk triggering process which would be necessary to enforce model tailoring or level of detail constraints on the effort as a whole (For example, Sussman correctly points out in the analysis of his own HACKER program that one of the main drawbacks to his approach is that it is very hard for the program to get the big picture of what it is doing. This is mainly because all of the expert knowledge is stored in separately triggered chunks which virtually come and go as they please (i.e., in accordance with the triggering that happens to arise during the processing of a particular problem)--see the conclusions in [Sussman]). Also there is the issue that if the chunks are allowed to operate on their own initiative (to maintain the desired locality), the problem of how to end up with an efficient problem-solving control structure becomes formidable.

In focussing, features of the input are not extracted to trigger chunks. Instead,

the piece of knowledge base which is suspected of being the model for the input is subjected to transformations until it is put into a form in which it can be matched to the input. These transformations are not applied in a phase-oriented refinement process, but rather are directed by two factors: differences between the piece of knowledge base and the piece of input, and the needs of the current modelling effort.

For example, as we saw in Chapter III, Alfred has three types of focussing transformations:

(1) Structural transformations,

(2) Transforming generics,

(3) Using alternate elaborations.

Let's examine each of these separately in order to understand how the focussing style of connection works.

Structural transformations are known ways to change from one expression of a piece of knowledge to a different expression of that same knowledge. Actually, the new expression will not represent exactly the same knowledge as the old one; it will only be "acceptably similar". In any focussing methodology, it is very important to have a good standard of what an acceptably similar transformation is at any time during the problem-solving effort. In Alfred, this notion is tied very closely to that of "the needs of the modelling effort"--that is, the restrictions posed by the trend. If, for example, the modelling effort were currently examining the effects of a flow containing A, B, and C on other parts of the model, and the A, B, C flow were represented as

(DETERMINED-BY B (INFLUX A) (OUTFLUX C)),

it would be "acceptable" to use the transformation which converts this construct into

(FLOW A B C).

That is, it's okay to think of this DETERMINED-BY as a FLOW in order to match it to a piece of input (which is presumably described as a flow whose object is "A")--but only because the modelling effort is currently interested only in the overall flow properties of A, B, and C. In general, this transformation implies a loss of information (i.e., the fact that B and C are decoupled by A); but now this loss is acceptable. We will see in Chapter VII that this "acceptability" restriction is enforced by not allowing structural transformations which introduce a more general formulation of a construct than the formulation that is in the TOWARD or PURPOSE of the trend (i.e., if the TOWARD or PURPOSE contained a DETERMINED-BY relationship between A, B, and C, the transformation would not be allowed; otherwise it would be).

As another example, it is possible, again with loss of information, to transform

(DETERMINED-BY A B C) or (DETERMINED-FROM A B C)

into

(DEPENDS-ON A B) and (DEPENDS-ON A C).

This is acceptable as long as the modelling effort is worried only about these single dependencies, i.e., is not worrying about the group aspects of the interdependency of A, B, and C.

The point that I am trying to make here is that focussing can work only if it is known what can be done to a piece of the model at various stages of the problem-solving effort without throwing the whole effort off track: otherwise the focussing process could simply wander off in any direction--it would become an exhaustive trial and error procedure. In practical terms, this means that the needs of the problem-solving effort must be expressible at every step of the process (which is the purpose served by the theme, trend, edge mechanism in Alfred).

This also holds true for transformation of generics. Here, however, the use of the generic itself poses a constraint on what can be used for focussing. As I said, earlier, generics are place-holders. Their presence indicates that the modelling effort is only thinking about a particular piece of the model as a DELAY, or SMOOTH, or (CAUSE), or (CHANGE)--i.e., nothing more specific than that, but only that. In other words, DELAY can be transformed into any tsructure which represents a DELAY, but that structure must indeed represent a DELAY. This means that the focussing apparatus has license to express the generic concept in any of the ways attached to it, as long as the suggested transformation does not conflict in some other way with the needs of the modelling effort. For example, SMOOTH can be transformed into a structure of the form

$$\text{smoothed quantity} = \frac{\text{quantity}_t - \text{quantity}_{t-1}}{\text{averaging time}},$$

as long as the "quantities" and "averaging time" (which may itself be a function) don't violate level of detail or other trend restrictions. DELAY turns into either a first- or third-order delay function similar in form to the SMOOTH above. (CHANGE) and (CAUSE) stand for practically any action. They are differentiated because they are used in different ways by the PURPOSE part of the trend, as we will see in the next chapter. For example, if the model piece under consideration were

```
(ACT-ON (NUMBER-OF (PRODUCT) (DECREASED))
        (CAUSE))
```

(i.e., "some action is reducing the number of products"), and the input construct were

```
(ACT-ON (NUMBER-OF (ORANGES) (DECREASED))
        (SPOIL))
```

(i.e., "spoilage is reducing the number of oranges"), (CAUSE) could be transformed to (SPOIL) to make the match. The importance of this is that (CAUSE) would then be identified with "spoilage" for any piece of the model interested in

(ACT-ON (NUMBER-OF (PRODUCT) (DECREASED))
(CAUSE)).

That is, other focussing transformations or reasoning processes which were dealing simply with (CAUSE) would be restricted to dealing with (SPOIL). Since (SPOIL) is more specialized than (CAUSE) (e.g., it can't be an act of the firm), this may have important consequences in restricting the modelling activity necessary to prevent or explain the reasons behind the (CAUSE).

I brought up this example to show the other prerequisite for using the focussing style of transformation: not only does its control require that much be known about the current needs of the modelling process at any time, but also, once a focussing transformation has been made, there must be some way to carry through the ramifications of that transformation throughout the rest of the modelling effort. In other words, the use of a focussing transformation, even if well motivated by and well within the restrictions of the current modelling effort, is going to have a specific influence on later parts of the modelling effort. Unless that influence can be easily transmitted to those parts of the model which are affected, focussing will be very hard to use.

The last kind of focussing is the most common and the most straightforward. It involves the substitution of a known alternative (i.e., an elaboration) for a stated abstract model construct. As before, the only tricky part of this is knowing what substitutions are allowable at any given point in the modelling effort--that is, what substitutions will not cause the modelling effort to digress from its current task. We have already seen a substitution of this sort in Chapter I, in which a backlog-dependent variant for determining the number of people needed by the firm was substituted for a more general term of the hiring policy. There will be more such focussing by elaboration in Chapter VIII. For now, suffice it to say that this kind of focussing needs the same kind of restriction- and consequence-monitoring environment as the other kinds--even more so, since, unrestricted, it is a completely open search-and-substitute procedure.

There are several other characteristics which are necessary in an problem-solving environment in which focussing is to be used. The first thing that we can note is that it is rather difficult to use the focussing approach in a distributed control structure. In Alfred, the connection effort is implemented by a single transformation procedure which

always starts with a specific piece of structure that it knows a great deal about (the given chunk of knowledge base) and always works toward a specific goal (the piece of input under consideration). The motivation for and restrictions on transformations are conveyed to the focussing apparatus through a single control procedure, not a number of independent chunks which are separately responsible for making their needs and constraints known. Furthermore, and perhaps more important, once a focussing decision is made, it is transferred to the rest of the model by a single procedure which sets up and focusses all of the chunks. In a distributed environment, the triggering mechanism of each frame would have to take special cognizance of each constraint as it was added, and as it was changed over time.

Another requirement for the use of focussing is that in order to ensure that the problem-solver will be able to handle the eventual connection that is made (i.e., the chunk as it is transformed to apply to the specific input), it is necessary to guarantee that the single transformation procedure always creates legal entities (i.e., entities that can be used by the problem-solving parts of the system). Again, this requires good knowledge of the model--a strong notion of legality for a particular phase of modelling operations. This means that the model must be expressible enough and self-consistent enough to deduce whether legality at a particular phase of model operation guarantees solvability when the model is finished. But this is hardly surprising: this is what expertise is all about.

The final requirement for focussing is that it be easy to access and transmit information from the whole knowledge base, including those chunks that have *already* been connected to pieces of the input. The focussing procedure must have this information in order to tailor its chunk transformation procedures to the problem description at hand. The theme, trend, edge mechanism provides the needed facility in Alfred.

If these requirements can be met, connection via a focussing procedure offers an important advantage over other approaches: it provides an organization for all of the expert's matching tricks (the "abstracters" described earlier). That is, instead of homogenizing these tricks into input transformation phases (a la approach (1)) or using them to provide the control structure (a la approach (2)), this approach allows them to be simply attached as passive information to the knowledge base chunks they concern--the focussing procedure will use them when it needs to.

The major disadvantage of this approach is that it is extremely difficult to recover from "errors" (in the sense discussed in association with the frame approach). That is, if the transformation procedure finds that it is on the wrong track, it is extremely hard for it to try something different while retaining the correct part of what it has done so far. This is because the choice of each individual transformation along the way was made in accordance with accumulated knowledge base and input information which is changed in the course of doing additional transformations. Also, since the connection knowledge of the focussing procedure is indeed procedural, it is rather hard to understand and use: it is easier to do than to know about.

It seems to me that the choice between triggering and focussing depends to a large extent on the nature of the knowledge base of the expert system. The chunk-based triggering approach is best suited to environments in which there exists a direct association between a chunk of the knowledge base and a piece of the input (i.e., there *is* a right chunk, even though it may be hard to find); in which the program must often use particular details of the input piece in order to choose the right chunk out of a group of similar, almost-right chunks; in which the major operating mode of "assume a particular chunk for the moment, but be ready to change your mind in response to additional input information" requires that the mechanism for giving up an old hypothesis and taking up a new one be efficient and preserve what has been learned so far; and in which the expert's matching tricks are plentiful enough and well-structured enough to be used as a control structure for the problem-solving process. On the other hand, the focussing approach seems to be adapted to environments in which it is fairly clear in most cases which chunk is going to apply to a piece of input, but in which it takes a great deal of work to actually apply it. This implies that the expert problem-solving methodology must be one in which a great deal is known *about* the knowledge in the knowledge base so that the focussing procedure can take full advantage of the program's accumulated knowledge at every stage of the connection effort. Also, if the expert's matching tricks don't cover the whole domain well or are poorly organized (i.e., the expert simply has a "bag" of tricks), this approach can still be used: whatever tricks there are are just stored with the knowledge chunk to which they apply.

It should now be clear why the recognitionists use a triggering approach to connection: their generalization-keyed models have exactly the ideal characteristics for expert methods that can use chunk-based connection. For example, note how well the

above description fits the approach to medical diagnosis seen in [Rubin]. On the other hand, as discussed in Chapter II, with the reformulation method there is no guarantee that there is a "right" chunk in the sense implied by the use of triggering. That is, in a reformulation environment, the knowledge base is sufficiently abstract that it is quite possible that nothing in the knowledge base can be associated directly with the input, even though there really is a chunk that applies to that piece of input. This also implies something about the reformulation expert's tricks: although he might have a number for each individual chunk of the knowledge base, he is unlikely to have ones which suggest one chunk over another. This is because the chunks are sufficiently abstract that either one of them *could* be right, depending on more global knowledge about the problem and the knowledge base. Thus, it is very hard to piece together a control structure out of matching tricks in a reformulation domain.

The focussing approach can handle the problems caused by the abstract nature of these chunks because it incorporates a procedure which is designed to transform a chunk that isn't right "yet" into an associatable chunk. There are also other characteristics of the reformulation environment which make it amenable to this approach. For one thing, its well-structured abstract model provides the external means for suggesting the "almost right" chunk to be transformed: whatever chunk fits in with the chunks that have been processed so far, or completes a needed section of the model. Second, the structure and abstractness of the abstract model also imply that there is no great need for the connection process to recover from "failures": the expert usually knows that a particular chunk *has* to be associated to make the whole modelling effort work; if he can't manage to somehow interpret that very abstract chunk in terms of the input he has at hand, he usually has to scrap the entire modelling effort (for that theory), not just the individual connection attempt. Finally, since the model is so abstract, it is extremely important for the program to be able to use what it has learned about the input so far to tailor the transformation process to the input at hand, i.e., to give it the "feel" of the problem so that it can be better at setting up model chunks for association.

It therefore seems that the focussing approach is the clear choice for an expert program which is to work in a reformulation environment, and, indeed, it is the method that is used in Alfred. However, it is not the whole story for Alfred's connection effort. It is now time for me to back off the absolutist stance taken so far in this chapter and explain how all three of the approaches discussed above are combined in Alfred's actual

connection process. We will see that although Alfred *emphasizes* the focussing approach, it must use elements of each of the other approaches in order to implement an efficient total connection process. (I think that any expert system would find that a combination of approaches would be best suited to implementing its particular connection process.)

In the first place, Alfred's connection process begins with a quick input-driven "canonicalization" phase which sweeps through the entire parser output, cleaning up, reformatting, substituting standard synonyms, and in some cases ignoring features of the input (as described earlier). This is all done before the modelling process even sees the input--so that the chunk transformation procedure will never have to worry about absolutely extraneous details and variations of wording in the input. Just as with other examples of the input-driven approach, this canonicalization phase is not directed toward specific pieces of input or knowledge base; it is a "filter" through which all of the input is passed. The system can get away with using this technique here without having to worry about the thorough overall planning and possible input rigidity mentioned earlier because the context is so limited and because the filter is totally syntactic. For example, as I said in Chapter III, the "matching" activity of the canonicalization phase is limited to direct synonym substitution. The whole idea is that other canonicalization operations should be similarly limited to those in which *no information is lost.* This is in contrast to focussing operations, in which information is always lost during transformation--but in ways that are acceptable to the needs of the modelling effort. Therefore, canonicalization is sort of the booster stage for focussing: a necessary one, I think, to keep the number of abstracters and focussing transformations within manageable limits.

After this input-driven start-up phase, the program actually begins its modelling effort--but with something very much like chunk-triggering. We saw in Chapter III that Alfred's first task is to find the symptom the client wishes it to work on. This starts with what is fundamentally a recognition problem: finding the "suspect" actions and effects out of which the symptoms are built. For this effort, Alfred uses what is basically a frame approach in which the abstracters attached to (SUSPECT-ACTIONS) and (SUSPECT-EFFECTS) are keyed to generalizations of the kinds of suspect actions and effects the program knows about. Some aspect of these "frames" will be the right associate for any actual suspect action or effect seen in practice; the problem is to choose between various similar ones to establish a connection. Once this connection is made for each of the suspect entities in the problem description, Alfred goes about its modelling effort in

earnest, actually formulating a symptom, deciding which of its theories is applicable, and modelling the problem in terms of that theory. This is all accomplished, as indicated earlier, via the focussing methodology discussed here (and in Chapter III and again in Chapter VII).

Finally, there is actually another brief chunk-triggered connection effort at the end of Alfred's problem-solving activity. This occurs when the problem has been completely modelled and Alfred must decide *precisely* what is wrong with the system under consideration by choosing between several given (very similar) alternatives. Clearly, this is another diagnosis-type problem which is best handled in a frame environment.

Thus, we can see that in any real expert system, various approaches will be needed to solve the whole connection problem. When there is a need for broad, uniform, basically syntactic processing of an entire structure, an input-driven phase is the answer. For the inevitable recognition problems that arise in any kind of expert problem-solving, chunk-triggering is the most efficient. And, for extensive in-depth modelling of uncertain input, I will try to show in the next couple of chapters that a focussing connection methodology is the most useful and natural.

Chapter VII
All About Themes, Trends, and Edges

The key to implementing a connection mechanism via the focussing approach discussed in the last chapter is having a good way to continuously adapt the chunk transformation process to the particular problem at hand as the modelling effort goes along. The basis of this mechanism in Alfred is the theme, trend, edge scheme mentioned earlier. We will see in this chapter that this scheme is useful not only for adapting the transformation procedure, but also for controlling the modelling effort of the program.

Theme, trend, and edge information is extracted from abstract model chunks *after* they have been connected to pieces of the problem-description. It is then used by the setup mechanism to tailor subsequent chunks to the problem. The setup mechanism also essentially provides the control structure of the modelling effort by deciding which chunks to set up when. To make these setup decisions, the program again must refer to the themes, trends, and edges, which in this context serve as a kind of database for the control structure. In the following sections I will discuss each of the three elements of the theme, trend, edge mechanism separately, explaining how features are extracted for each element and how they are used to control program activities.

Section 1 Themes

As the name implies, themes embody basic goals of the problem-solving effort. In Alfred, the theme for each problem is always based on the symptom which the program finds in the problem description (i.e., the presenting symptom). The basic content of the theme is a description of how the symptom should be treated by the program. That is, should it be further investigated, eliminated, its cause eliminated, or what? This description is in the form of goals (in the "classic" sense used in many problem-solving programs) which are constructed around the actual symptom as it is recognized by the program. The exact form of the goal structure depends on particular characteristics of the symptom. For example, given the symptom

```
(NUMBER-OF (PRODUCT (RETURNED))
           (MANY)),
```

the program would see that it was dealing with an undesirable effect whose *cause* should be found and eliminated (i.e., the program will not be satisfied with merely eliminating the symptom of the problem--it's very easy to stop products from being returned, but it doesn't help matters much!). This would give rise to the theme

```
((ACCOUNT-FOR (CAUSE-OF
    (NUMBER-OF (PRODUCT (RETURNED))
               (MANY))))
 (ELIMINATE (CAUSE-OF
    (NUMBER-OF (PRODUCT (RETURNED))
               (MANY)))));
```

straightforward enough. Sometimes it is necessary only to eliminate a symptom, not its cause, to solve a problem. Specifically, when the symptom refers directly to an action or sector of the firm, it is enough to eliminate the stated malfunction. Thus, for

```
(STATE-OF ((FIRM) (POLICY) (ORDER (BACKLOGGED)))
          (UNMANAGEABLE)),
```

the program will be satisfied with accounting for and eliminating the unmanageability of the policy, not the cause of it (which the program wouldn't know how to find anyway).

ACCOUNT-FOR and ELIMINATE are the only kinds of themes implemented in Alfred. They are usually used together, as above, to schedule the program's activities for handling the usual kind of case problem it is faced with, but may be used separately if required information is already present or if a different task is being set. That is, if the symptom has been sufficiently accounted for by the user, only ELIMINATE is necessary. If the user is only asking why a certain effect occurs, or why the program did a particular thing, ACCOUNT-FOR is used by itself.

The theme information in ACCOUNT-FOR and ELIMINATE is used in two basic ways. First of all, the theme defines the problem-solving effort for any particular case.

That is, it tells the program what to apply its theory to, why it's applying the theory (i.e., what it's trying to accomplish), and what to do after the theory has been applied. Second, the theme directs the problem-solving effort by establishing bounds for the setting of trends and by directly suggesting chunks which must be set up and modelled. In particular, the theme makes sure that the program is always trying to solve the client's problem, not just modelling to be modelling.

Themes are used in a very straightforward way to control the setup and use of theories. ACCOUNT-FOR is a signal to gather together all of the possible relevant theories and to begin the theory selection effort. The only way that the program can *account-for* a symptom is to explain it in terms of a theory in the abstract model. ACCOUNT-FOR therefore means that the program must model the symptom in terms of the problem definition statement of one of its theories. We will see the process of theory gathering and selection in Chapter VIII in connection with Alfred's application of the workforce need theory to the symptom

```
(ACT-ON (PEOPLE (EMPLOYED (WORKING)))
        ((OR (HIRE) (LAYOFF))
           MODIFICATION (FREQUENT))).
```

The process is nothing elaborate--just the gathering of all theories which have a problem definition whose object is the same as (or "similar to" in the sense that will be described in section 3.2) the object of the symptom (in this case, (PEOPLE (EMPLOYED (WORKING)))), followed by the setting up of a trend to model the symptom. The theme is used only to control the setting up of this effort--it does not participate in the details of the modelling effort. (This is the province of the trend.) The ACCOUNT-FOR theme is considered complete when the selected theory has been successfully modelled, as determined by the completion of all trends and "OPEN's" (see below).

The ELIMINATE theme is used in a precisely similar way to signal the "diagnosis and treatment" phase of the program's activity. After the symptom has been adequately modelled in terms of a theory, the program must find a treatment for it and apply that treatment to the modelled system. ELIMINATE is the signal for the program to look down the treatments attached to the theory to find one that is applicable and effective. In order

to determine this, it must usually try to apply the selected treatment to the modelled system. All of this is signalled by ELIMINATE. Again, this process is shown in Chapter VIII. ELIMINATE is considered complete when a treatment has been successfully applied to the modelled system.

Thus, as far as theory handling goes, themes act very much like goals in the traditional sense. They establish the task of the problem-solving effort and act to set up the appropriate subgoals. Other than that, they are simply placeholders, "pushed" to indicate that the program is involved in a certain activity, and "popped" when the activity is completed. Nonetheless, this "goal" aspect of the theme is extremely important because it introduces the symptom as a guide for problem-solving. The theme expresses the symptom as the goal of the modelling effort, thus establishing the symptom information as "that which is to be modelled". The modelling effort eventually goes far beyond the original symptom modelling to encompass all relevant aspects of the firm, but the effort is started by and directed toward the symptom itself. This is accomplished by setting the appropriate trends to fulfill the theme.

The other aspect of the theme is its direct control over the setup of abstract model chunks via the OPEN property. OPEN is used to advertise unmodelled pieces of the abstract model which must eventually be modelled in order to complete the case under consideration.

The source of these OPEN pieces is the ABBREVIATION mechanism of the trend which was discussed in Chapter III. Its use, we remember, is to hold pieces of the abstract model which were previously brought up, but were not modelled in full detail--for any of the several possible reasons discussed in section 3.4. As far as the theme is concerned, the only interesting aspect of ABBREVIATION's is that they sometimes hold pieces of the abstract model which are in sectors that are not in the close interaction group being handled by the current trend. This comes about when a flow or function in the close interaction group under consideration contains a factor which is part of another sector (e.g., the "number of people needed by the firm" which is part of the workforce sector depends on factors which are in the production sector--these two sectors form separate close interaction groups). If an external factor like this is considered by the program but not fully modelled, the whole modelling mechanism must be informed, since the decisions about how that piece should be modelled (level of detail, etc.) depend on needs

which are outside the hegemony of the trend. Furthermore, once a piece of a sector appears in the modelled system, the sector containing that piece must be fully modelled at some level of detail in order for the system to be complete--not have undetermined elements or flows that lead nowhere. Sectors which are introduced by having pieces modelled in other trends are considered to have been "opened" by that trend, and must be closed (i.e., modelled, even if at a very high level of detail) before the modelled system can be analyzed and (especially) simulated. In other words, open sectors are a model-wide consideration, and must therefore be recorded as part of the theme, viz., on the OPEN property. Since open sectors must be modelled, the theme is capable of providing direct influence on what model chunks are to be set up, especially in the later phases of the modelling effort when mop up operations are done. We will see this happen in Chapter VIII.

Section 2 Edges

If the theme is the most general possible guideline for the modelling mechanism, the edges are the most specific. They express the boundary conditions that must be met by the next piece to be put into the modelled system. That is, they represent the aspect of the piece just put in which the next piece must deal with. For example, consider the flow

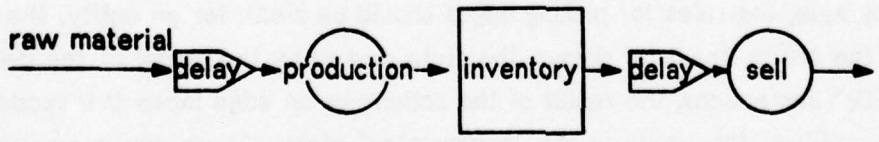


Figure 25. Production flow to show how edges are chosen

and assume that the program is modelling it left to right, step by step (the usual case in such matters). Suppose that "raw material" is modelled as (PIG-IRON). Since this is what the rest of the flow must deal with, "(PIG-IRON)" becomes the edge, i.e., the thing that "delay" must somehow handle. If for some reason "raw material" could not be modelled (and it was known or assumed that some raw material really existed), the edge would simply be (RAW-MATERIAL). If it was discovered that there definitely was no raw material

(say the firm was engaged in typesetting like Dominion), the edge would be NIL. This would have immediate consequences, since when the setup mechanism went to setup "delay", it would discover the null edge, and realize that the delay would be dealing with nothing. This would mean that "delay" should not be set up. The setup mechanism would immediately check to see if that delay was a necessary characteristic of the flow. If it was, an "edge lapse" would be noted (this affects the trends--see below); if it was not, the modelling would proceed with "production".

Assuming that (PIG-IRON) or (RAW-MATERIAL) was the edge, though, "delay" would be set up to handle it. Suppose that the delay was modelled as (STORAGE). The new edge would be (PIG-IRON (STORED))--the effect of the delay and the thing that production would have to deal with. Again, (PIG-IRON (DELAYED)) would be possible. This time, however, if the firm were found not to have that delay, the edge would still be (PIG-IRON)--"delay" had no effect and production deals simply with (PIG-IRON). Coming now to the modelling of "production", the modelling mechanism must go off and discover what the firm makes. This could lead to a whole separate modelling effort (production is often modelled as a flow itself), but whatever the complexity of the production model, the result of "production", say (WIDGET), would be the edge when the modelling mechanism returns to the flow of figure 25. The edge after "inventory" would (always) be (WIDGET (INVENTORIED)), unless "inventory" turned out to be unmodellable, in which case the whole flow would collapse.

At any rate, the rules for picking edges should be clear: for an entity, the model of the entity or the entity itself; for delays, the state caused by the model of the delay, or simply "(DELAYED)"; for actions, the result of the action--or an edge lapse if unmodellable; for accumulation entities, the entity in the "accumulated" state. There are a couple more things to be said about edge selection, though.

One is that the above examples and rules cover only what happens when a *flow* is being modelled. Although this is what is going on during 90 percent of the modelling effort, it isn't everything. In particular, non-flow modelling occurs when the symptom and diagnosis are being modelled, and when a piece of model is being examined under the special PURPOSE constraints discussed in the next section. Since these pieces usually do not become part of the model of the firm itself, I have not determined what the concept of "edge" means in these cases. Therefore, when these pieces are incorporated into the

modelled system, there is no edge (this is different from a null edge), and the modelling effort simply does without edge information.

The second notable point concerns the way edges are used by the modelling mechanism. I have already mentioned that a null edge causes a quick check to be made for an "edge lapse", that is, a problem which usually causes some kind of change of tack in the modelling effort. The same thing happens whenever the next piece to be set up somehow can't make use of the edge. Something is definitely wrong here, and it is the edge that is considered sacrosanct since it is directly due to the modelling of a previous piece (which depended on the previous edge, etc). Therefore, if the abstract model piece under consideration cannot be set up in such a way that it can use the existing edge, the modelling mechanism reports that it "can't set up a piece" to the trend-changing mechanism discussed in the next section.

Finally, I mentioned earlier that the program checks the model for consistency (using the knowledge attached to FLOW given in V-1.2) as it incorporates each piece into the modelled system. This is all done with edges. If an inconsistency is detected, an edge lapse like the ones above (i.e., "can't set up a piece") is recorded. For example, even though the (STORAGE) delay in the above example could accept (PIG-IRON (PROCESSED)) just as well as (PIG-IRON), this edge would be unacceptable in the given flow because a new entity cannot be created in a flow without a causing action (from rules (3) and (5) under FLOW). Also, (WIDGET) is okay under rule (4) of FLOW because it is similar to (in the sense of section 3.2) (PIG-IRON (PROCESSED)). If the modelling of "production" had come back with (REQUISITION), an edge lapse would have occurred.

Edges, then, ensure the continuity of the model. The setup mechanism, as the final guarantee that a chunk is appropriate, looks at the edge of the contiguous piece of modelled system and sets up the chunk to specifically make use of that edge. Since the edge was the result of previous model-tailoring via the trend, the program can be sure that the chunk it is setting up fits in smoothly with the trend and maintains the grain and integrity of the model. When a conflict arises, i.e., when no chunk can be set to fulfill the need specified by an edge and still conform to the other constraints and goals of the theme and trend, it is usually a sign that the trend should be changed. But this gets into the issue of trends...

Section 3 Trend

I said earlier that the Alfred program is good at tailoring the abstract model to the problem at hand and at modelling at various levels of detail. I have also said that it works by "focussing" pieces of the abstract model onto pieces of the input. Almost all of the information for doing these things is kept in the trend. The trend is the repository of the non-local but not totally global information about modelled pieces which makes the above kind of processing possible. As I said in Chapter III, this is just the information that pieces within a "close interaction group" of the modelled system have in common.

Of course, "have in common" is a rather misleading way to put it; part of the reason that the chunks inside the group have the information in common is that the way in which they were set up was heavily influenced by the information already in the trend. As chunks are associated with pieces of the input, some of the new problem description information the associations bring to the modelled system is abstracted out and added to the trend in order to refine and perpetuate it. In this way the information in the trend gets better and better for tailoring model pieces to the problem description. The purpose of this section is to describe the way in which the relevant features of the newly associated pieces are abstracted out and later used to set up other chunks of the abstract model for the modelling process.

In the following subsections I will discuss these abstraction and use issues individually for each of the kinds of information that can appear in the trend. I will then conclude the section with a discussion of how trends are changed. This will cover the "focussing" and "database for the control structure" uses of trend information.

3.1 The TOWARD constraint

The major purpose of the trend is to constrain and guide the setup of the abstract model chunks over which it has hegemony; the embodiment of this function is the TOWARD constraint of the trend. In fact, the TOWARD is the seminal trend-*defining* feature--it expresses the basis of the trend, what got it started in the first place. The TOWARD is used in three ways to constrain and guide the modelling effort:

<> It restricts the use of abstracters.

<> It restricts the use of focussing transformations.

<> It preserves alterations from earlier model tailoring to be used in further modelling of the close interaction group it controls.

Before getting into the details of how the TOWARD is used to do this, it is necessary to show where TOWARD constraints come from, that is, which features of the modelled system can become the TOWARD of a trend.

There are three sources of TOWARD constraints. The first is the symptom discovered by the symptom-finding mechanism and are placed in the theme. Since the initial trend of the modelling effort is toward finding the theory which applies to the symptom, the TOWARD is abstracted from that symptom. It says, in effect, "move toward making what you're modelling (in this case, the problem definition of a possibly applicable theory) relevant to the symptom". This is the primary controlling mechanism for making sure that the modelling effort doesn't go wandering off on its own and try to solve more than the given problem of the firm under consideration. (We'll see how this is enforced later in this section).

An example of a symptom-based TOWARD constraint (from the Future Electronics fragment shown in figure 13, Chapter IV) would be

```
(FIRM-CHARACTERISTIC
      (NUMBER-OF (PRODUCT (RETURNED)) (MANY))
        REASON (DEFECTIVE)
        DURATION ((PERIOD) MODIFICATION (SOME))).
```

It would be used for directing the initial modelling effort toward finding something that could deal with the characteristic of periodic production of defectives. Furthermore, it would not allow the modelling effort to work on anything else for a while (e.g., it shouldn't go off and figure out how products are made, yet).

The second major source of TOWARD constraints is the basic ideas of sectors

and the major flows within the model. That is, when the modelling effort has reached the stage of actually reformulating the problem description in terms of the model, trends will be toward modelling particular sections of the abstract model of the firm. The TOWARD always expresses the program's *current* version of the basic idea of the sector (i.e., it includes any model tailoring that might have been done to any part of the basic idea in a previous modelling effort). Since this is what is used to restrict the modelling activity of the program on that sector at any stage of the problem-solving process, this kind of TOWARD is one of the major feedback links between the setup mechanism and the modelled system.

We saw a basic idea based TOWARD in Chapter III--the basic idea of the workforce sector--

```
(DETERMINED-BY (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
    (INFLUX (DELAY1 (ACT-ON (PEOPLE)
                            ((FIRM) (HIRE)))))
    (OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                             ((FIRM) (LAYOFF))))))).
```

This is typical of this sort of TOWARD: the major flow of the sector being modelled. Of course, as more is learned about the characteristics of the problem at hand, and these characteristics affect the flow model which is used for the TOWARD, the TOWARD will be changed. That is, if the workforce sector must be examined again, the TOWARD set up is not the same DETERMINED-BY structure shown above, but that structure as *tailored* to the problem as far as the current state of knowledge allows (perhaps the DELAY's will be modelled, or (LAYOFF) will have been changed to (FIRE), etc.). This is how model tailoring information is transmitted in Alfred, and also how trend restrictions are kept as up to date as possible as modelling progresses.

Note the shift in emphasis between the two kinds of TOWARD's mentioned above. The first was based almost completely on the needs of the problem description. This reflects the program's initial bias toward letting the problem description establish the bounds of the modelling effort. The second kind of TOWARD reflects the needs of the abstract model, as influenced (more and more as time goes on) by the needs of the modelled system. This reflects Alfred's propensity to see things in terms of its model.

The final source of TOWARD constraints is the diagnosis of the actual cause of the symptom which is made after the modelled system is complete. It reflects total bias toward the needs of the modelled system: the problem description and the abstract model mean nothing now (everything the program needs from them is already in the modelled system).

For example, one of the causes of personnel fluctuation that the program can diagnose in the modelled system (and thus possibly make into a TOWARD) is

```
((STATE-OF (EFFICIENCY-OF (PEOPLE (EMPLOYED (TRAINING))))
           (DECREASED))
 MODIFICATION (UNACCOUNTED-FOR)).
```

That is, the firm has failed to take into account the reduced efficiency associated with trainees--they don't know their jobs as well as the average employee, regular employees must use part of their time to train them, etc. (This happens to be the diagnosis for the Future case, of which part (not the part that suggests this diagnosis!) was shown in figure 13.) The idea here is that the program must realize that the close interaction group covered by this trend is being modelled in order to find a solution which addresses that *particular* diagnosis. As usual, the TOWARD is there to make sure that it does that and nothing else.

So much for where TOWARD constraints can come from. After they have been selected, from whatever source, the important issue becomes how they are actually *used* to constrain and guide. The first two aspects of the TOWARD's management of the modelling effort, limiting the use of abstracters and limiting the use of focussing transformations, work via the same *restriction* mechanism. We saw earlier that the connection process used by the program recursively goes through the chunk under consideration, trying to match each piece (in the technical sense of "piece" given in Table 1 of Chapter III) with some input construct. This connection process uses the abstracters attached to the piece under consideration and the focussing transformations known to the focussing mechanism. The choice of abstracters and focussing transformations is limited by the following rule: if the TOWARD contains a piece which is more restrictive than the piece under consideration, only the abstracters and elaborations of the more restrictive piece may be used in the connection process.

The comparative restrictiveness of any two pieces can be determined on the following basis:

(1) A piece that is in a particular state is more restrictive than that piece without state specification. (E.g., (PEOPLE (EMPLOYED (WORKING))) is more restrictive than (PEOPLE (EMPLOYED)), which is more restrictive than (PEOPLE); (STATE-OF (PEOPLE (EMPLOYED)) (FLUCTUATING)) is more restrictive than (PEOPLE (EMPLOYED)).)

(2) A more fully specified version of a piece is more restrictive than the less specified version, where specified means that "NIL" has been replaced by another model construct. (E.g., (NUMBER-OF (PEOPLE (EMPLOYED)) (DECREASED)) is more restrictive than (NUMBER-OF (PEOPLE (EMPLOYED)) NIL); (ACT-ON (PEOPLE (EMPLOYED)) (TRAIN)) is more restrictive than (ACT-ON NIL (TRAIN)).)

(3) A piece which takes the place of a generic is more restrictive than that generic. (E.g., (ACT-ON (PEOPLE (EMPLOYED)) (TRAIN)) is more restrictive then DELAY; (ACT-ON (PEOPLE) (HIRE)) is more restrictive than (ACT-ON (PEOPLE) (CHANGE)).)

(4) A characteristic of a piece is more restrictive than that piece. (E.g., (EFFICIENCY-OF (PEOPLE (EMPLOYED (TRAINING)) (DECREASED)) is more restrictive than (PEOPLE (EMPLOYED (TRAINING))).)

(5) An elaboration of a piece is more restrictive than that piece. (E.g., the (TIMES (NUMBER-OF (PEOPLE (EMPLOYED)) NIL) (ATTRITION)) elaboration of ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL) MODIFICATION (DESIRED EXTRA)) is more restrictive than ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL) MODIFICATION (DESIRED EXTRA)) itself.)

The use of these restriction rules to limit the choice of available abstracters prevents the association of abstract model concepts to input concepts which are not of

interest in the current close interaction group. Remember that the rule is that if the TOWARD contains a more restrictive version of the piece under consideration, only the abstracters of the more restrictive version may be used. For example, suppose that the TOWARD of a particular trend were

```
((NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) (DECREASED))
    MODIFICATION (UNDESIRED UNACCOUNTED-FOR));
```

that is, "the on-the-job workforce is decreasing in an undesirable and unexpected way" (this is an example of a symptom-based TOWARD). Further suppose that during the modelling of this close interaction group, the piece

```
(ACT-ON (NUMBER-OF (PEOPLE (EMPLOYED)) (DECREASED))
    (QUIT))
```

("quitting decreases the number of employees") were set up in order to investigate the cause of the symptom. Only the abstracters of this piece which could referred to (PEOPLE (EMPLOYED (WORKING))), *not* simply (PEOPLE (EMPLOYED)), would be allowed to be used. (As a practical matter, since pieces like (ACT-ON (NUMBER-OF (PEOPLE (EMPLOYED)) (DECREASED)) (QUIT)) don't have abstracters of their own, but use the abstracters of the pieces they contain, this restriction would come into force when the (PEOPLE (EMPLOYED)) piece of this piece is set up; at this time, only the abstracters of (PEOPLE (EMPLOYED (WORKING))) would be allowed to come into play.) This is an important consideration if the input discusses several kinds of quitting. If, for example, it said that 5 percent of the trainees quit because the training is too tough for them and 50 percent of the actual workforce quits because of injury (say that the case is talking about the "Green Berets") the program must be sure to get the right information for this close interaction group. Both trainees and actual workforce can be matched by the abstracters of (PEOPLE (EMPLOYED)). Therefore, it is only the restriction imposed by the TOWARD which limits the connection process to using the abstracters of (PEOPLE (EMPLOYED (WORKING))) and accessing only the correct information. This is the contribution of the trend information to the selection problem discussed earlier.

The use of the restriction rules to limit the focussing transformations that can be applied not only helps with the task of making sure that only associations of interest to the trend are made, but also prevents the modelling effort from going off on a tangent (with respect to the current trend). As I said earlier, the association of one piece of the abstract model causes the setting of others. The major vehicle of this process is the use of focussing transformations to introduce elaborations (and definitions of generics). Therefore, the focussing mechanism must be carefully limited to setting up only pieces which are of direct interest to the current close interaction group--otherwise the modelling effort would turn into a depth-first search through the space of all of the elaborations of the pieces it came in contact with for any given problem. The restriction mechanism prevents this by limiting the focussing mechanism to setting up only elaborations and generic definitions which are *more restrictive* than the current version in the TOWARD. This means that if a generic has already been defined in the TOWARD, or an elaboration has already been chosen for an action or entity in the TOWARD, the focussing mechanism may not set up a different or less restrictive definition or elaboration. It can only use the existing one or set up a more restrictive one. This is extremely important in limiting Alfred's problem-solving activity.

For example, suppose that in a trend whose TOWARD contained (PRODUCTIVITY) (like the close interaction group which is used to model the number of people desired by the firm--see V-1.3), (PRODUCTIVITY) had been modelled as a (CONSTANT). That is, the (CONSTANT) elaboration of (PRODUCTIVITY) had been used to model the fact that productivity was given as a constant value in the problem description, as it is, say, in Dominion (1000 lines per day). Now suppose further that this close interaction group were being reexamined in order to find out why (PRODUCTIVITY) was decreasing: i.e., the piece

(NUMBER-OF (PRODUCTIVITY) (DECREASED))

was set up. There are many elaborations of (PRODUCTIVITY) which make this an interesting question--in other words, a hard modelling effort. However, since (PRODUCTIVITY) has already been modelled as a (CONSTANT) in the TOWARD, the program is restricted to using only that elaboration (there are none more restrictive). Under these conditions, the cause-effect tracing routines (see the next section) which answer

questions like the above one have a very easy task. For a *constant* productivity to have decreased, either the user must have explicitly lowered it, or the program itself must have explicitly lowered it. Thus, the effect of the restriction mechanism on focussing is to limit the program's problem-solving work to only that which is relevant to the current problem at hand--a use of "feedback through the modelled system".

The control of structural transformations by the TOWARD is quite straightforward and does not even merit a "restriction rule". Structural transformations are always set up to lose information. That is, one may transform a structure into another structure which does not preserve the information content of the original one, but not vice versa. Thus, it is possible to change a DETERMINED-BY into a group of DEPENDS-ON's, but not vice versa (this would in fact require some kind of "proof" that a given group of DEPENDS-ON's included all of the relevant factors for determining whatever had to be determined--something Alfred can't handle). The restriction policy is that unless this less information-filled version is in the TOWARD, the transformation may not be used. Thus, if the TOWARD contained (DEPENDS-ON A B) and a piece (DETERMINED-BY A B C) was under consideration, it would be okay to use the DETERMINED-BY-to-DEPENDS-ON transformation, otherwise not. Similarly, unless (FLOW A B C) were in the TOWARD, the transformation from (DETERMINED-BY B (INFLUX A) (OUTFLUX C)) would be disallowed. This is in order to assure that the information lost in making the transformation was not of interest to the close interaction group.

Finally, note that focussing transformations are restricted in the other ways discussed in Chapter III: until "object matching" is successful, only transformations which change the object of a chunk may be used, and later, when there is a specific input piece under consideration, focussing transformations are suggested on the basis of piece by piece comparison with the input. These may be seen as restrictions which balance the limitations set by the TOWARD: they make sure that the focussing process first tries to set up pieces for which there is a possible association already in the problem description. Of course, it may be desirable to set up a piece for which there is no association in the problem description, but since its association involves asking the user for information, it should be saved for last.

The third way that the TOWARD is used differs from the constraining sense shown above: it conveys information already discovered about the model back to the

modelling process. The example of the (CONSTANT) elaboration of (PRODUCTIVITY) above was an example of such feedback. We will see another one in the next chapter, in which it is disovered in modelling the workforce sector of Dominion that the entity (PEOPLE (EMPLOYED)) should be broken into three entities representing people who have just been hired, people who have been trained, and people who have been laid off but haven't left yet. Since (PEOPLE (EMPLOYED)) is part of the TOWARD of the workforce sector close interaction group, this learned information is carried with the TOWARD (as a MODIFICATION of (PEOPLE (EMPLOYED)) *in that TOWARD*--i.e., nowhere else, since this information, may be inapplicable outside of the close interaction group) every time that sector is set up for reexamination. As discussed in Chapter I, this is one of the major model tailoring mechanisms of the program.

We can see, then, that the TOWARD is that part of the trend which is used to make sure that the modelling activities of the program are restricted to the needs of the close interaction group (*for each close interaction group in the model*). It is also used to preserve information that has been discovered about that close interaction group during modelling. Putting these two uses together, the TOWARD can be seen as limiting the problem-solving activities for the theory selection, sector modelling, and solution finding activities of the program using the best current notion of the relevant symptom, sector, or diagnosis known at the time that the TOWARD is set up. The TOWARD is indeed the fundamental feature of the trend; but it is too confining. The modelling mechanism needs more freedom of action and more kinds of information to pursue its activities. This additional capability is provided by the other features of the trend, as described in the subsections below.

3.2 The PURPOSE constraint

It often happens that in order to model within the restrictions of the TOWARD of a trend, the modelling effort must go off on a tangent--usually to check whether a focussing transformation being suggested is valid. The TOWARD constraint is much too restrictive to allow this kind of divergence from the mainstream effort, so these temporary wanderings are handled under the aegis of special PURPOSE constraints. The PURPOSE is essentially a "subgoal" which is used very much like (and in addition to) the TOWARD of

the trend. It does not take the place of the TOWARD, but it does remove the TOWARD restriction on abstracter usage and replace it with a restriction (of exactly the same kind, i.e., using the same restriction rules given above) based on the PURPOSE entity. The PURPOSE can also bypass the level of detail restrictions of the trend, as we will see in the next subsection. It is used in all other ways like a TOWARD; i.e., it is used to choose abstracters and focussing transformations. However, the entities that go into PURPOSE constraints are very different from those that are in TOWARD's; the two constraints express very different kinds of things--which is why the PURPOSE should not be thought of as "another TOWARD". The PURPOSE constraint basically expresses the reason why the program is going off on a tangent.

In the current Alfred program, there are four possible reasons for temporarily diverging from the trend:

<> MODEL: Sometimes, in order to progress with the modelling effort of the trend, it is necessary to take a step back and take a deeper look at how things are done or how they connect up in the firm in question. This almost always takes the form of modelling (at a high (i.e., gross) level of detail) the sector which contains the things "under consideration". Such an action is in fact the default when the modelling effort becomes stalled (unless it is prevented by something else in the theme or trend). These little side excursions of the modelling effort, which clearly diverge from the TOWARD of the trend, are constrained by PURPOSE. For example, suppose the program were trying to model

```
(STATE-OF (NUMBER-OF ((CUSTOMER) (ORDER)) NIL)
          (DECREASING))
```

and stalled, i.e., had no piece of model which could fit the edges and could be set up. It would then decide to find out about customer ordering, that is, find out the factors that the customers use in making their ordering decision. To do this, it would model the CUSTOMER-ORDERING sector, starting by setting up the basic idea and putting

```
(PURPOSE (MODEL
  (STATE-OF (NUMBER-OF ((CUSTOMER) (ORDER)) NIL)
            (DECREASING))))
```

on the trend. That is to say, "my purpose in modelling the CUSTOMER-ORDERING sector is to find out why customer orders are decreasing".

<> SHOW-CAUSE: In modelling the firm, and especially in finding a solution to the diagnosed problem of the firm, the program often likes to change from dealing with an effect or characteristic to dealing with the action that gives rise to that effect or characteristic. This is because the program can do a lot more with actions: they are the "variables" of the system--they are what the manager can change. Also, the program knows a lot more about actions in terms of more elaborations, knowledge about various policies which can control the action, and the knowledge attached to generic actions like (CHANGE). At any rate, the task of finding the cause of an effect comes up often enough to deserve its own PURPOSE "subgoal". For example, if the program were dealing with the effect

```
(NUMBER-OF (PEOPLE (EMPLOYED (TRAINING))) 0)
```

and instead needed the action which caused the effect, it would put

```
(PURPOSE (SHOW-CAUSE
  (NUMBER-OF (PEOPLE (EMPLOYED (TRAINING))) 0)))
```

on the trend and start looking for a causing action. For example, the program would begin by looking back through the FLOW for the workforce sector (see V-1.2 and figure 17) looking for possible causes. Suppose that it finds

```
((ACT-ON ((PEOPLE) MODIFICATION (EXPERIENCED))
         (HIRE))
 MODIFICATION (ONLY)).
```

Now, knowing from the characteristic information attached to (PEOPLE (EMPLOYED (TRAINING))) (also in V-1.2) that trainees are necessarily inexperienced, Alfred can use the piece of deductive knowledge attached to SHOW-CAUSE (which is part of SHOW-CAUSE--see below) that says "if you ACT-ON only *x*, then you don't ACT-ON *not x*" to realize that this (HIRE) action is the required cause of the "lack of trainees" effect. Once this is done, the PURPOSE is considered to be satisfied, the constraint is dropped, and the trend continues under its TOWARD, now having this new model of the cause of the effect.

<> SIMILAR?: As mentioned in connection with symptom-finding, the program must sometimes decide whether two actions or entities are similar enough to be considered the same under the circumstances. "Under the circumstances" here translates into "as restricted by the current theme, trend, and edges", so that it is already built into the the TOWARD, as discussed above, before the PURPOSE constraint is set up. Nonetheless, there is usually still a considerable amount of effort necessary to show similarity; certainly enough to go off on a tangent. Thus, if the program finds the association

```
(DETERMINED-BY ((NUMBER-OF (PRODUCT (RETURNED)) NIL)
                           PER (WEEK)
                           MODIFICATION (AVERAGE))
   (TIMES ((NUMBER-OF (PRODUCT) NIL)
                 PER (WEEK)
                 MODIFICATION (AVERAGE))
          .05)),
```

(i.e., that 5 percent of the products produced per week are returned) when it has already modelled

```
((NUMBER-OF (PRODUCT (RETURNED)) 5000.)
      PER (DAY)),
```

it would have to discover whether or not these two associations said the same thing. It would do this by investigating the daily rate of production, doing the multiplications, etc., first putting

```
(PURPOSE (SIMILAR
   (DETERMINED-BY ((NUMBER-OF (PRODUCT (RETURNED)) NIL)
                        PER (WEEK)
                        MODIFICATION (AVERAGE))
      (TIMES ((NUMBER-OF (PRODUCT) NIL)
                  PER (WEEK)
                  MODIFICATION (AVERAGE))
             .05))))
```

on the trend to control the effort. Clearly, a SIMILAR? PURPOSE could cause a MODEL or other kind of PURPOSE to be started up, etc.

<> CONTRADICTION?: It very often happens that in associating a section of the abstract model, especially at a high level of detail, it is necessary to show only that there is no contradiction between the suggested piece of model and the statements of the problem description in order to have the piece accepted into the modelled system. This is handled under a CONTRADICTION? PURPOSE constraint. For example, suppose the program wants to use

```
((ACT-ON (GEAR) (PRODUCE))
     AGENT (FIRM))
```

as a model for how gears get into the firm. Assuming that it doesn't want to know about this process in any more detail, it is allowed to simply assure itself that there is no reason for not believing it. To do this, it adds

```
(PURPOSE (CONTRADICTION?
  ((ACT-ON (GEAR) (PRODUCE))
      AGENT (FIRM))))
```

to the trend, and uses the deductive knowledge stored with CONTRADICTION? to look for contrary evidence in the problem description. If it finds, say

```
((ACT-ON (GEAR) (PURCHASE))
      AGENT (FIRM)),
```

it has found such a contradiction; the PURPOSE is satisfied, and the earlier suggestion of *producing* gears must be discarded.

As I said at the beginning of this subsection, PURPOSE constraints work very much like TOWARD's as far as restrictions on abstracters and transformations go. However, there are a couple of essential features of PURPOSE which make its use quite different from that of TOWARD.

The most important difference, alluded to twice above, is that each of the PURPOSE constraints has special "how to" knowledge attached to it. This is knowledge which is more general than the abstracter knowledge attached to entities and actions, and even more general than the knowledge attached to modelling entities. It is essentially information on how to use these other kinds of knowledge for the special purpose of the PURPOSE. For example, attached to MODEL is stuff like "start with the basic idea of a sector, and then work through the major flows"; "in the absence of level of detail information (the usual case for PURPOSE's), the default is to model at the highest known level of detail"; etc. SHOW-CAUSE has things like "use the cause-effect knowledge attached to FLOW's to trace causes" and "if x *is* y then x can be said to cause y" (we'll see this little number used in Chapter VIII). SIMILAR? basically says to first check for "reasonable" congruence of structure between the objects in question (i.e., both objects should be ACT-ON's, but they needn't have the same modification, or a DEPENDS-ON can be similar to an ACT-ON if the ACT-ON is a substructure of the DEPENDS-ON, etc.), then

to check for identity of elements, and finally, to check for "necessary characteristics" etc. information of the non-identical elements (if two things have enough of the same characteristics they can be considered similar). CONTRADICTORY? works in much the same way, only this time, clearly, looking for conflict rather than identity. Therefore, each PURPOSE constraint has its own special repertoire of ways to go about modelling in order to achieve its subgoal.

The other difference between PURPOSE and TOWARD is that PURPOSE's can "fail" easily. That is, the PURPOSE constraint is designed as a trial effort which is simply "popped" if it does not come to fruition. It is not used for many-step modelling efforts, and its consequences can not be incorporated into the modelled system until the whole PURPOSE is finished. A PURPOSE can be thought of as a specialized problem-solving procedure which is given a particular argument (the piece of model specified in the PURPOSE statement) to work on. If the procedure completes on that argument, the PURPOSE is considered to be accomplished. If it does not, the PURPOSE has failed and its consequences are ignored. This is very different from the TOWARD, which is a data structure which is used to guide a much more general modelling procedure. If the modelling procedure cannot continue given the constraint imposed by the TOWARD, the whole trend is considered to have failed. In other words, something is wrong with the current modelling effort, and a new direction (i.e., a different working constraint) must be chosen if Alfred is to proceed with the problem.

PURPOSE is therefore the closest thing to a problem-solving goal used by the program. It is not only the only real device for "probing" for a solution, but is also the repository for the program's deductive procedures for using the characteristics and modelling knowledge stored under the individual actions, entities, and modelling entities of the abstract model. (The deductive procedures themselves are independent of the particular actions, entities, and modelling entities involved--they just know how to *use* the model-specific information).

3.3 LEVEL information

Since closely interacting pieces of the abstract model are presumably all at the

same level of detail (else they couldn't talk to each other), level of detail direction is an obvious candidate for inclusion in the trend. That is, once a level has been decided on for a certain piece of abstract model, that same level should be used to set up future pieces for association--as long as it is "valid". Since we are assuming that it will be valid, i.e., appropriate, across an interaction group, and have no reason to believe that it will be valid outside of the group, it is reasonable to propagate the level via the trend. This is the purpose of the LEVEL property of the trend.

First of all, though, just what is a level of detail and how can it be expressed? Level of detail can be stated only in terms of model structure. It is the grain of differentiation at which a particular aspect of the problem is being modelled. That is, should the production/distribution part of the firm be thought of as simply "production"?; as the filling of orders?; as the assembly, inventorying, and shipment of products in response to orders?; as the 17 parallel assembly lines and 4 special order job shops needed for widget production, an inventory department with special storage and depreciation rules, and a fleet of 43 blimps and 2 handcars for widget distribution?; etc. As the above sentence shows, the only *real* way to talk about level detail is in terms of specific pieces of the model. However, this isn't much help in terms of relating the "level" of one piece to the "level" of another, e.g., is (HIRE) at the same level as (PEOPLE)? (PEOPLE (EMPLOYED (WORKING)))? (ORDER (BACKLOGGED))?

This is where the structure of the model comes in. Individual pieces are related to each other in terms of level via structural relationships in the model. For our abstract model, this means that the level of detail of actions and entities is related to the level of other actions and entities via *modelling entities.* The way the LEVEL part of the trend information is abstracted from the modelled system and used to set up new pieces is as follows...

First, "seed" actions and entities are chosen to establish the level at which the interaction group is being modelled. Seed levels are chosen from the TOWARD constraint, since it represents the defining aspect of the trend. These seed levels are simply the actions and entities of the TOWARD constraint, recorded under the LEVEL property of the trend. As each new piece of abstract model within the abstract model is set up, it is checked against the LEVEL information. If its actions and entities match (i.e., are identical to) those of the LEVEL, the piece is okay. If not, an attempt is made to relate the level of

the actions and entities of the prospective piece to the level of the actions and entities under LEVEL.

This is done by tracing through the modelling entities in which both a prospective entity or action and an entity or action in the LEVEL appear, according to the following rules:

(1) All elements of a FLOW are considered to be at the same level.

(2) All elements of a DETERMINED-BY are considered to be at the same level.

(3) The object of an ACT-ON is considered to be at the same level as its action.

(4) The result of an ACT-ON is considered to be at the same level as the ACT-ON.

(5) REASON's and other ancillary aspects (e.g., AGENT, EFFICIENCY-OF, etc.) of a modelling entity are *not* at the same level as its major components.

(6) An elaboration of a modelling entity is *not* at the same level as the entity itself.

(7) DEPENDS-ON and DETERMINED-FROM provide no level of detail information one way or the other.

(8) If the actions and entities being checked are not related by any modelling entities, there is no level of detail information one way or the other.

If the actions and entities being checked are found to be at the same level as those in the LEVEL property, the setup process continues right along. If they are found to be at a different level, elaborations of the prospective piece are checked. If no elaboration at the same level as the LEVEL information can be found, the trend-changing mechanism is brought in (see subsection below). If there is no level of detail information (i.e., rules (7) and (8) above), the setup process is allowed to continue, but a note is made that if anything goes

wrong later, the trend-changing mechanism should consider that the divergence began at that point. All new actions and entities found to be at the same level as the seed actions and entities are added to LEVEL so that the simple identity check will prove more and more fruitful as the trend continues.

This is the basic level of detail propagation mechanism in the program. It is complicated slightly by a couple of other factors. First of all, level of detail information is considered to be theory-dependent. That is, when checking for a relationship between actions and entities that are in question, the modelling entities of the theory being used are checked first. This is because in some cases the theory could view the relationship between certain actions or entities differently than does the abstract model as a whole. Furthermore, relationships within a sector are checked before those of the theory, because it is *possible* that there could be a difference (it hasn't happened yet, but just in case). Finally, level of detail information is strictly differentiated with respect to the area or agent of the actions or entities (i.e., (PEOPLE (EMPLOYED)) in the firm is not considered identical with respect to level to (PEOPLE (EMPLOYED)) in a competitor, etc.)

There is one more point. When PURPOSE constraints are used, LEVEL information can temporarily be bypassed. The idea is that since the PURPOSE can involve activities which are not within the interaction group, there is no reason to enforce level of detail restrictions. If it is possible for the PURPOSE to stay within the constraints imposed by the LEVEL property, it does so (i.e., LEVEL is always checked). However, if it is not, it is not a cause for failure (as it would be in the TOWARD).

LEVEL information thus provides an additional check on the modelling activities of the program, acting more or less as an extension to TOWARD.

3.4 ABBREVIATION

Whenever the program has the opportunity to model a piece that has been set up, but decides not to, it records that fact in the trend under ABBREVIATION. There are two possible reasons for deciding not to model something once it has been set up: it is contraindicated by level of detail considerations, or there is not enough information about it to model it. These two reasons give rise to three kinds of ABBREVIATION's.

Suppose that the program were trying to model the (ORDER-FILLING) activity of the firm and found that it was at the same level as the LEVEL information. This is an indication that (ORDER-FILLING) should not be modelled in further detail at this time. The program only has to check to make sure that there is no contradiction of (ORDER-FILLING) in the problem description. If it ascertains that there is indeed some kind of order filling activity going on, it can go ahead and incorporate (ORDER-FILLING) as (ORDER-FILLING) in the modelled system. However, it must also place

(ABBREVIATION ((ORDER-FILLING) (SLUFFED))

on the trend, a reminder that the program had the opportunity to model (ORDER-FILLING), but did not, *without even looking to see whether modelling was possible.* This is what SLUFFED means. When the program sees a SLUFFED ABBREVIATION, it knows that it should be careful about making deductions based on it (i.e., it should model it in more detail before it concludes anything about it). It also knows that the modelling effort should start from scratch: i.e., it wasn't even attempted before, so very little is known about the ABBREVIATE'd object.

Now suppose that the program, in checking for a possible contradiction with (ORDER-FILLING), discovered that no order filling activity for the firm was mentioned in the problem description. There would be three courses of action...The program could chuck out the model because something is wrong, ask the user for more information, or simply assume that there really is an order filling activity even though it's not mentioned, and move on. Clearly, the choice between these alternatives depends on what it is that's being modelled and its importance to the theory under consideration. For a major activity like (ORDER-FILLING) (which is a "necessary characteristic" of (FIRM)), it seems reasonable to simply assume that it's there and ask the user about it later, if necessary (no point bothering the user until it's really necessary). Of course, the program must remember that this model of (ORDER-FILLING) is just speculation on its part, so it records

(ABBREVIATION ((ORDER-FILLING) (ASSUMED)))

on the trend in case it ever has to remember where (ORDER-FILLING) came from. (The

user might ask, or the program may want to see whether the (ORDER-FILLING) model can be depended upon to be the basis of an important decision, e.g., theory selection.) Also, in some cases, ASSUMED ABBREVIATION's are used to trigger dialogue with the user. That is, ASSUMED tells the program that the only way it can find out about (ORDER-FILLING) (say, to really confirm its existence, or to model it in further detail) is to ask the user.

While this assumption idea is all right for something like (ORDER-FILLING), it would not be appropriate for any of the less certain characteristics of the problem description. For example, suppose that the program were wondering whether the firm had a (TRAINING) activity. Now, although many firms do some form of training, others simply do not have enough training activity (or what they do have is not sufficiently differentiated from other activities) to warrent modelling as a separate (TRAINING) action. Therefore, the program will ask the user about the firm's training activity immediately rather than making any assumptions. If he confirms the existence of (TRAINING) (via a simple "yes"), all well and good. However, if he does not, this fact is recorded in the trend as

(ABBREVIATION ((TRAINING) (BLOCKED)))

so that its ramifications are immediately apparent to the close interaction group--references to (TRAINING) will be edited out of model pieces being set up, etc.--and so that the user will not be bothered about training again. Note that this is definitely a trend level consideration: lack of (TRAINING) could be perfectly all right in modelling one interaction group and fatal to modelling another. Thus, the BLOCKED ABBREVIATION would be used quite differently in the two trends. This is why it, as well as the other ABBREVIATION's, goes on the trend rather than elsewhere.

On the other hand, if the trend is completed (i.e., TOWARD and PURPOSE's satisfied) and there are still outstanding ABBREVIATION's, it is a matter for more general concern. Specifically, if the ABBREVIATION's refer to sectors which were not modelled during the trend, they are placed in the theme under the OPEN category. As we saw above, this means that they must be modelled in order to achieve a "complete" modelled system. The idea here is that as long as the ABBREVIATION's are in the sector that the trend is modelling, the program can adopt the attitude that the trend knows its own business best as far as what it can afford not to model. However, for pieces that go

beyond that sector, other trends should be informed about what has been skipped and why. Therefore, when trends are changed, a simple check of the remaining ABBREVIATION's is made to see which ones are "open", that is, are in, and thus introduce the need for, unmodelled sectors. These are simply placed with the other "to be modelled" pieces on the theme for more global consideration.

ABBREVIATION's, then, are the signposts of the trend, showing where exploration was stopped and where dead ends were found. This information is used to edit the pieces being set up and to provide clues of what work needs to be done if the pieces covered by the trend are ever reexamined in more detail. In this capacity, ABBREVIATION's are often used to cue particular questions to the user.

3.5 Changing trends

Anyone with any experience in problem-solving will be able to predict that the trend-changing process will not be a smooth and easy one. Changing the trend implies the decision to stop pursuing one line of investigation and begin another. This decision can be made because the current line is considered to be "wrong", i.e., it is ineffectual for modelling the problem, or because the current line is felt to be "finished", i.e., it has contributed all it can. In either case, something must decide that the current line of investigation is no longer worth pursuing. This is a very difficult decision for a human expert, much less an expert program. We know that people frequently make mistakes in this area; in fact, no one always makes the right decision, and few people (even few experts) can be said to make the right decision most of the time for a hard problem. Judgement is complicated by the fact that the decision is often highly debatable, even in retrospect. This is all by way of introducing the fact that we cannot expect a clean, direct trend-changing mechanism which is demonstrably right all of the time. The mechanism used in the Alfred program represents a theory of when to change directions in the problem solving effort which arises naturally (I think) from the other devices described in this chapter. There is no way to say that it is the best possible trend-change methodology for the program; there isn't even enough evidence to say whether it is particularly good or bad. All I can do is present the rules I use and say that they seem to work pretty well in the situations I have tried them on. This business of when to shift the problem-solving attack is clearly an area for further research in expert system design.

Change of trend in Alfred is always keyed by (1) the program not having a piece to set up, or (2) the failure of the program to find an association for a piece it has set up. We have seen that the modelling effort works by setting up a structure of model pieces (such as the basic idea of a sector or a flow in the system) and then associating them with pieces of the problem description. The original structure and its immediate ramifications form the close interaction group which is monitored by the trend. As the modelling effort proceeds, there will come a time (barring other problems) when all of the pieces and all of their immediate ramifications have been set up and associated (actually, this usually means only about a dozen associations in all): nothing is left. Clearly, a change of trend is indicated.

Whether or not this is a "normal" completion (i.e., the trend is considered complete, the line of investigation has contributed all it can), is determined solely by whether or not the TOWARD constraint has been utilized in the modelling effort, that is, whether or not all of its pieces have been checked for restriction constraints. Remembering back to where TOWARD's can come from, we can see that this is really interesting for only two out of the three kinds of TOWARD's. For symptom-based or diagnosis-based TOWARD's, the "utilization" means, as it should, whether or not the symptom or diagnosis has determined the way the theory or solution (respectively) has been modelled and is going to be used. If it has not, the trend cannot be considered to be complete, but rather off the track. That is, the line of investigation has petered out without producing results. For abstract model based TOWARD's (i.e., those based on the basic ideas of sectors), this isn't an issue. The trends which contain these TOWARD's automatically utilize them: they work by modelling the object of the TOWARD itself step by step--the TOWARD is *the* piece of structure which is the basis of the interaction group. Thus, if a trend with a model-based TOWARD terminates because there are no more pieces to set up, it has terminated normally.

Trend change is also signalled by the fact that a piece that has been set up cannot be associated. This is always an abnormal termination of the trend, and can occur for any kind of TOWARD.

Thus, we can see that the conditions which signal trend change are quite clear cut. Also, the decision of whether or not the trend terminated "normally" is made to be straightforward. For trends which terminate normally, even the issue of where the new

trend is to come from is clear cut. The sole source of new trends after a normal termination is the OPEN property of the theme. That is, assuming for the moment that the modelling effort proceeds forward unfalteringly, we can imagine the initial trend being set up and completed, "opening" new sectors as it goes. These go on to the OPEN property of the theme and are used to start new trends (their basic ideas becoming the TOWARD's). These may in turn produce more OPEN's as still other sectors are touched. At any rate, when all of these OPEN's are finished off, the modelled system is complete.

The picture is not so rosy for abnormal terminations. Here the problem is not simply one of what to go off and do next in the modelling effort. It is more a matter of: first, what can be done to salvage the current trend; second, what alternate approaches are possible; and finally what must be done to recover from the effects of the mistake. To begin with, consider the case of abnormal termination by running out of things to set up (not having used the TOWARD). Here, nothing can be done to salvage the situation without changing the trend. What the program can do depends on the source of the TOWARD. For symptoms and diagnoses, an alternate symptom or diagnosis must be chosen. If there are none that apply to the problem, the program returns with "can't figure out what the symptom is" or "can't figure out what's wrong". If the user can supply a new symptom, the program tries to model it, and the effort begins again from the beginning. If not, it quits. If the program "can't figure out what's wrong", it always quits. For TOWARD's whose source is a piece of the abstract model, a different expression of the TOWARD (i.e., a different elaboration of the element in the TOWARD) is sought. If there are none, the program fails.

For abnormal termination because of inability to model a piece which has been set up, the situation may be salvageable without changing the trend. The program asks the user a question based on the point it is stuck on. For example "I can't figure out how your hiring policy works" or "Where do gears come from?". (The questions are always quite simple template-type queries based on the piece that is giving trouble. That is, the program always asks how x works if x is an action, or where x comes from if x is an entity.) If the user answers the question, the answer is parsed and canonicalized, and added to the problem description. The program then tries to model the setup piece again. If it is now modellable, the trend continues; if not, the trend is not salvageable, and a new trend must be started up.

But where is that new trend to come from? As discussed in this section, the

setting up of pieces within a trend is affected by LEVEL and TOWARD information, and the use of abstracters to find associations for these pieces is affected by the TOWARD. Thus, even small changes in these aspects of the trend can change a piece from being unmodellable to being modellable, even with the same set of abstracters. The program proceeds as follows. First, if there were any pieces which were set up with no LEVEL information (see subsection above), the program tries to reset them at a new level of detail, chosen to be lower (i.e., more finely grained) by default. All trend work done after the offending piece was set up is discarded, and the trend is begun again with the new LEVEL information. If this fails, the next higher level of detail is chosen for the offending piece. If this too fails, or if there were no other levels, or if setting the piece at a new level conflicted with other LEVEL information, the program tries something else. "Something else" can only be changing the TOWARD. As before, for TOWARD's drawn from symptoms and diagnoses, the only choice is to try a different symptom or diagnosis. Again, if this fails, a question is asked or the whole effort fails.

However, the case of an abstract-model-based TOWARD is somewhat more interesting. In all of the cases, inability to make an association with a setup piece means that a particular group of abstracters has failed to find what it was looking for in the problem description. Similarly, in all of the cases, this could be due to the fact that the scope of the abstracters was restricted by the TOWARD. Now, however, because the TOWARD is abstract model based, there may be other elaborations of the TOWARD which preserve the "idea" of the trend, but change the restrictions on abstracters. The program therefore looks for an alternate TOWARD which is not as restrictive as the previous one from the point of view of the piece under consideration. For example, a TOWARD such as

```
(RATIO (SMOOTH (DIFFERENCE ((NUMBER-OF (PRODUCT (INVENTORIED)) NIL)
                                MODIFICATION (DESIRED))
                            (NUMBER-OF (PRODUCT (INVENTORIED)) NIL)))
       (PRODUCTIVITY))
```

(taken from ((PEOPLE (EMPLOYED)) MODIFICATION (DESIRED EXTRA)) in V-1.3) would restrict the abstracters of the piece under consideration to only looking at (PRODUCT (INVENTORIED)) rather than, say, (PRODUCT). If the (PRODUCT) abstracters were the ones that were having trouble, the program would look for an alternative to the above TOWARD (a different elaboration of ((PEOPLE (EMPLOYED)) MODIFICATION (DESIRED EXTRA))) which removed the restriction, e.g.,

```
(RATIO (SMOOTH (DIFFERENCE ((NUMBER-OF (PRODUCT) NIL)
                            MODIFICATION (EXPECTED SEASONAL))
                           (NUMBER-OF (PRODUCT) NIL)))
       (PRODUCTIVITY)),
```

or had a different restriction, as in

```
(RATIO (SMOOTH (NUMBER-OF (PRODUCT (BACKLOGGED)) NIL)
       (PRODUCTIVITY))).
```

If it can't find one, the modelling effort fails as before. If it can, it tries the abstracters under the new restriction. If they succeed, the new elaboration is substituted for the old TOWARD and the trend is restarted. Naturally, there is no guarantee that the trend will not bomb out again (perhaps it won't even get as far as the other one did), but at least this gives the program a life, and ensures that it will try all reasonable possibilities before giving up.

Now, suppose, for any of the above cases, that the trend must be discarded and a new one started up. I must emphasize that Alfred has no mechanism for saying "what's still right" from before when it restarts the trend[1]. Indeed, it is no easy matter to figure out what's still right, because, as I pointed out in the chapter on connection, the effects of the old trend and the implications of the new one are *implicit* in the structure of the interaction group that was set up under the old trend. I have not yet found a way to separate out what in the trend caused what in the interaction group, so total restarts are necessary. However, since, as I said, interaction groups are only on the order of a dozen associations, this is not really a terribly serious problem. On the other hand, it would be nice "some day" to have enough knowledge about the theme, trend, edge mechanism to be able to remodel only what is necessary, answer questions about which trend constraints caused certain features of the modelled system to be represented as they are, etc.

[1]The exception is the "no level information" case, in which the trend may be restarted from the point of the "error". But any change to the TOWARD implies restarting the whole trend from the beginning.

Returning to the here and now, though, I will leave this general description of themes, trends, and edges and go on to show how the methodology is actually applied in a detailed example from the Alfred program.

Chapter VIII
A Detailed Example

This chapter contains a step-by-step account of Alfred's attack on the Dominion case shown in figure 3 (straight from [Jarmain]) in Chapter I and figure 26 (actual input form) on the next page.

I will use the early stages of the modelling process for this case to show how chunks are chosen for processing, what changes are made to the chunk when it is set up, how the various stores of knowledge discussed in Chapter V are used, etc. As the method becomes clearer, I will move along at a more business-like clip.

The initial chunk set up by the system is (always)

```
(DETERMINED-FROM (SYMPTOM (UNDER-CONSIDERATION))
        (SUSPECT-ACTIONS)
        (SUSPECT-EFFECTS)).
```

As we saw in Chapter V, the DETERMINED-FROM implies that the program should not look for a direct association with (SYMPTOM (UNDER-CONSIDERATION)) in the problem description, but should expect it to be derivable from (SUSPECT-ACTIONS) and (SUSPECT-EFFECTS). However, in this case, the first abstracter attached to the chunk overrides this general modelling knowledge and proceeds immediately to trying to find a piece of problem description which corresponds to (SYMPTOM (UNDER-CONSIDERATION)). This is because it sometimes happens that the manager simply says "and this is the symptom you should work on..."; it's at least worth checking for (of course the manager can be wrong, etc.). So far then, the "DETERMINED-FROM" structure of the chunk means essentially nothing, and the program is involved in a straightforward association attempt.

1 The firm's name is "Dominion Typesetting Inc.". 2 The industry's name is "typesetting". 3 The industry is very competitive. 4 Customer ordering policy is: consider (usually) both price and estimated delivery time. 5 The industry's most important cost is labor. 6 Dominion's labor use policy is: maximize the efficiency of workforce use (maintain the product price at a competitive level). 7 The industry backlog policy is: maintain a customer order backlog (if the minimum workload is not sufficient, some workforce is idle (sometimes)). 8 Dominion often hires or lays-off workforce, in spite of this backlog policy.

9 The industry's workload has random day-to-day fluctuations. 10 Hiring policy is: if the customer order backlog is large, hire more workforce (customers might transfer orders to a competitor). 11 Layoff policy is: if the customer order backlog is small, lay-off some workforce (if minimum workload is not sufficient, some workforce is idle (perhaps)). 12 If even a small portion of the workforce is idle, money is lost because of the present highly competitive prices.

13 The industry knows relatively little about customers. 14 If order backlog is much above 2 weeks, sales are decreased (usually). 15 The industry's average order backlog is 2 weeks. 16 If order backlog is much below 2 weeks, sales are increased (usually). 17 Sales are relatively constant. 18 Industry sales are relatively constant. 19 Sales have unexplained bad periods and good periods.

20 The workforce is union (primarily). 21 Most of the workforce are "typesetter"s. 22 Each typesetter produces 1000 "line"s/day (usually). 23 Training takes 5 days (average). 24 Average notice period for layoff is about 5 days. 25 A substantial number of unemployed typesetters are always available. 26 Dominion's personnel policy is: avoid indiscriminate hiring and layoff (the union might demand higher wages). 27 If the union demanded higher wages, Dominion would be in a bad position, because the competitors have different unions for typesetters.

28 Dominion's workforce is now 100 typesetters. 29 Dominion's output is 100000 lines/day. 30 The average daily order rate during the last 8 weeks is 100000 lines. 31 The unfilled order backlog is 1000000 lines. 32 Dominion's customer order filling delay is 2 work weeks because of the unfilled order backlog. 33 The industry's average customer order filling delay is 2 work weeks. 34 Dominion's workforce level was constant during the past several weeks. 35 Dominion's prices approximately equal competitor's prices. 36 Dominion's pricing policy is: maintain present price level during the next few months.

Figure 26. Actual program input form

This initial "find the symptom directly" abstracter fails for the Dominion problem, since the manager never does say what he thinks the symptom is (see figure 26). This being the case, the usual meaning of DETERMINED-FROM takes over, and Alfred decides to find (SUSPECT-ACTIONS) and (SUSPECT-EFFECTS) in order to determine a symptom. The program finds exactly one (SUSPECT-ACTION),

```
(ACT-ON (PEOPLE (EMPLOYED (WORKING)))
        ((OR (HIRE) (LAYOFF))
             MODIFICATION  (FREQUENT)))
```

(from sentence 8), and one (SUSPECT-EFFECT),

```
((FIRM) (SALES))
  (CHARACTERISTICS
    ((AND ((PERIOD) MODIFICATION (BAD))
          ((PERIOD) MODIFICATION (GOOD)))
      MODIFICATION (UNMANAGEABLE))),
```

that is, that the entity ((FIRM) (SALES)) has the characteristic (see Table 1, Chapter III) that the sales have unmanageable good and bad periods (from sentence 19). Actually, the order filling delay suggested by sentence 32 was considered as a possible (SUSPECT-EFFECT), but was rejected because of the information in sentence 33. As mentioned earlier, this involves the setting of a SIMILAR? PURPOSE in the trend and an investigation of the problem description. The check is so simple here, though, (because of the structural similarity of sentences 32 and 33 (see the actual case in figure 3 to make sure I'm not pulling a fast one)) that I will not show it here. Of these, the best symptom candidate, as discussed in V-2, is the "in spite of" action. The resulting association with (SYMPTOM (UNDER-CONSIDERATION)) is:

SYMPTOM

```
(SYMPTOM (UNDER-CONSIDERATION))=

(((IN-SPITE-OF) (ORDER (BACKLOGGED)))
   (MANAGER-DESCRIPTION
      (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
              ((OR (HIRE) (LAYOFF))
                 MODIFICATION  (FREQUENT)))


                AGENT    (FIRM (ASSUMED))
                EVIDENCE ((IN-SPITE-OF POLICY BACKLOG)
                          (ACTION: ((OR ((HIRE) NIL)
                          ((LAYS-OFF) NIL)) (OFTEN))
                          (OBJECT: (WORKFORCE))))))


   REASON  IN-SPITE-OF-POLICY
   SYMPTOM-SUPPORT  NIL
```

Note that the reason for thinking that this is a symptom is recorded (in case anybody asks) and that there is no other support for the symptom except the evidence of sentence 8.

When this association is incorporated into the modelled system, the following theme and trend are constructed:

```
THEME=
     ((ACCOUNT-FOR (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                       ((OR (HIRE) (LAYOFF))
                                            MODIFICATION (FREQUENT)))
                                        AGENT (FIRM))
                                      NIL))
     (ELIMINATE (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                    ((OR (HIRE) (LAYOFF))
                                         MODIFICATION (FREQUENT)))
                                     AGENT (FIRM))
                                   NIL)))


TREND
   CONSTRAINTS
      TOWARD (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                ((OR (HIRE) (LAYOFF))
                                     MODIFICATION (FREQUENT)))
                               AGENT (FIRM))
                             NIL)
   LEVEL
            FIRM-ACTION    (HIRE LAYOFF)
            FIRM-ENTITY    ((PEOPLE (EMPLOYED (WORKING)))).
```

The directive power of the symptom for the problem-solving effort is immediately realized in this theme and trend. The theme is in fact a direct call to investigate and dispose of the undesired action described in the symptom. The initial trend contains a constraint which limits the problem-solving activity to analyzing only the given piece of structure (again, the cause of the symptom, taken straight from the theme) until more information about the problem is known (it shouldn't go off and try to model the whole problem situation just yet). The rest of the trend is level of detail information which prevents the program from getting mixed up in the fine points of the problem at this stage. Thus, Alfred is limited both in scope and level of effort by its initial model of the presenting problem. Since the symptom is not actually a "physical" piece of the system being modelled, it produces no edge information to affect the rest of the modelling effort at the micro level.

Note throughout the following description the way the lead is passed between the problem description and the abstract model by the chunk setup process. Especially note the way in which the manager's description of the presenting problem heavily influences the theory selection effort but does not restrict the way in which the rest of the model is constructed (the rest of the model is in fact directed by TOWARD's which are based on sectors--i.e., which express the influence of the expert's model on the way the manager's problem should be handled).

So then, we are under the

```
(ACCOUNT-FOR
      (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                         ((OR (HIRE) (LAYOFF))
                               MODIFICATION (FREQUENT)))
                               AGENT (FIRM))
                 NIL))
```

theme, with no chunk currently under consideration. As discussed earlier, this is the program's signal to gather together all "conceivably relevant" theories which can attack the symptom which is currently expressed in the TOWARD of the initial trend. This gathering together is done by the "object matching" mechanism discussed in Chapter III. That is, all theories whose problem definition chunk has as its object (PEOPLE (EMPLOYED (WORKING)))--the object of the symptom--will be gathered together. As always, this is simply a way of marshalling all conceivably relevant knowledge of the program for the problem description (without committing much effort to the process). Next, Alfred will see if any of the theories thus collected really *are* relevant to the symptom under consideration. That is, it must determine if the manager's problem can be seen as one which one of the relevant theories can solve. If the problem definition of one of the theories can be made to match the symptom (by focussing transformations), that theory is considered to be applicable to the problem at hand. It is important to realize that, just as for the initial gathering process, the selection of a theory to fit a symptom is exactly the same as the connection of any other two pieces. The same focussing transformations and trend restrictions apply. Thus, it may be useful to refer to the flowcharts given in Chapter III and the details of TOWARD's and PURPOSE's given in Chapter VII as I go through this theory selection effort.

The "workforce-need" theory which, as explained earlier, deals with workforce fluctuations, is a member of the "conceivably relevant group" by virtue of its problem definition statement:

```
(CAUSE-OF
  (STATE-OF
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
      (FLUCTUATING))
  ((FIRM-ACTION)
            MODIFICATION  (UNDESIRED
                           UNACCOUNTED-FOR))).
```

That is, this particular theory is claiming to be able to work on any problem whose symptom is a fluctuating workforce level caused by an undesired, unexplained (i.e., there are no known constraints in the system which force the necessity of this particular action) action of the firm. This problem definition statement is "gathered" by the object matching mechanism described earlier. In fact, it takes a recursive application of the object matching procedure (as discussed in Chapter III) in order to see this as a valid candidate. That is, the problem definition statement of the workforce need theory is a candidate because (PEOPLE (EMPLOYED (WORKING))), the object of the object of the TOWARD, matches (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)[1], the object of the object of the problem definition chunk. The program can get away with this "hunting for the object" because, as mentioned earlier, the representation used in Alfred is so structured that the object always corresponds to "what the modelling entity is talking about". That is, it is what actions act on, what is determined in a DETERMINED-BY structure, the thing whose state is being highlighted by a STATE-OF, etc. Now, since chunks can have several layers of modelling entity structure, and since focussing transformations can change this modelling entity structure according to the needs of the close interaction group (as we saw in the previous chapter), it is necessary in gathering candidates to use only the "real" object of

[1]This is a match because NUMBER-OF with a third element of NIL is transparent with respect to the connection process. This is because in this configuration its use depends only on whether or not the program is considering the numerical aspect of a particular entity at a particular time. That is, it can be inserted at will by either the program or the user whenever the count of any entity is desired. Since (NUMBER-OF ... NIL) is so ubiquitous, and so nothing-meaning from the point of view of connection, it is ignored. If the third element *is* specified, the NUMBER-OF is making a statement about its second element, and is therefore of interest to the connection process (much in the same way STATE-OF is).

the chunk--the first *action* or *entity* in the object position. That is, both the TOWARD chunk and the problem definition chunk are considered to be "talking about" the same thing: (PEOPLE (EMPLOYED (WORKING))). This definition of object matching is deliberately not very discriminating--the idea is to collect all possible candidates; chunks that are really inapplicable are quickly weeded out by lack of abstracters or focussing transformations (see the flowchart in figure 12 of Chapter III).

Getting back to the theory selection effort for Dominion, the problem definition chunk shown above is the exact form in which it appears in the abstract model. As it is set up, Alfred checks theme, trend, and edge information to see if the chunk should be modified to fit the situation at hand. There is no edge so far, but the existing level of detail information in the trend has direct bearing on the chunk. A check of the actions and entities of the chunk to be set up against the LEVEL property of the trend shows that they are all either at the same level as the LEVEL actions and entities, or there is no level of detail comparison with those actions and entities. Actually, the only comparable entity is (PEOPLE (EMPLOYED (WORKING))), and it is seen to be at the right level by direct checking. This is hardly surprising at this early stage of the modelling effort. The candidate problem definition chunk is therefore setup as is, and the connection effort begins in earnest.

This means that the problem definition chunk is the piece under consideration, and must be matched with the input construct under consideration, which is the symptom from the TOWARD. This state of affairs, in which the TOWARD contains the piece from the problem description (after it has been modelled as a symptom, of course) and the abstract model piece (i.e., the problem definition) must be "gathered", is a natural consequence of using the symptom for the TOWARD. The decision to do this is of course based on the need to have the user's presenting symptom tailor the expert model. The point is that the connection mechanism is in no way altered by the fact that the TOWARD also happens to be the input piece: it poses restrictions on the connection mechanism just like any other TOWARD. In fact, the matching of a symptom to the problem definition of a theory is usually very efficient, because the limitations posed by the TOWARD are identical to those posed by the input piece under consideration.

The connection process for this chunk begins, as usual, with the selection of abstracters attached to the piece under consideration. They are passed through the

restriction mechanism of the TOWARD and then applied to the input piece under consideration to see if there is a match. In this case, there are no abstracters attached to the problem definition chunk. It is too big. Big chunks are very specific--they represent a very particular configuration of abstract model concepts. It is unreasonable to expect that there will be abstracters, specialized matching routines, for such a very specific concept structure. These big chunks are usually matched piece by piece using the abstracters attached to their various component pieces.

Since there are no abstracters to be applied, the connection process next tries focussing transformations. There are no transformations which can be applied (the two chunks under consideration are represented by the same modelling entity, and the problem definition chunk, again because it is so specific, has no elaborations attached to it). Therefore, the connection process recurses, and is applied to the first element of the chunk under consideration and the first element of the input piece under consideration. (If the chunk under consideration were not represented by the same modelling entity as the input piece, a structural transformation would have to take place at this point so that co-recursion down the two structures would make sense.) That is, the connection process is now trying to match

```
(STATE-OF (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
          (FLUCTUATING))
```

with

```
(ACT-ON (PEOPLE (EMPLOYED (WORKING)))
        ((OR (HIRE) (LAYOFF))
            MODIFICATION (FREQUENT))).
```

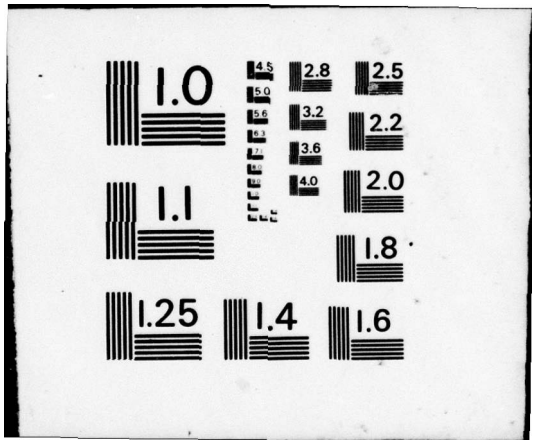
Again the connection process looks for abstracters under the piece under consideration. This time there are some. They pass the TOWARD restriction because there is no piece of the TOWARD which is more restrictive (indeed the (STATE-OF (PEOPLE (EMPLOYED (WORKING))) ...) in the piece under consideration is more restrictive than the (ACT-ON (PEOPLE (EMPLOYED (WORKING))) in the TOWARD). However, the

AD-A035 397 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
THE REFORMULATION MODEL OF EXPERTISE.(U)
DEC 76 W S MARK N00014-75-C-0661
UNCLASSIFIED MIT/LCS/TR-172 NL

3 OF 3
ADA035397

END

DATE
FILMED
3 - 77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

abstracters fail: there are no abstracters which recognize frequent hiring or layoff as a kind of fluctuating workforce. As we can see by the figure 12 flowchart, this means that it is time to look for focussing transformations in order to align the conceptual structures and try again.

There are no structural transformations which apply here, nor are there any generics. This leaves only the possibility of elaborations. Accordingly, the focussing mechanism looks for elaborations of

```
(STATE-OF (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
          (FLUCTUATING))
```

which will make it more similar to

```
(ACT-ON (PEOPLE (EMPLOYED (WORKING)))
        ((OR (HIRE) (LAYOFF))
            MODIFICATION (FREQUENT))).
```

In particular, elaborations of the form (ACT-ON (PEOPLE (EMPLOYED (WORKING))) ...) are looked for. There are none. Again, the focussing process is applied recursively on the piece under consideration. The object, (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL) is the same as the object of the piece under consideration (remember that (NUMBER-OF ... NIL) doesn't count), so the focussing mechanism doesn't even look for transformations here. However, (FLUCTUATING) does offer a possibility. Specifically, it has an elaboration of the form

```
(ACT-ON NIL
        ((CHANGE)
            MODIFICATION (REPETITIVE OSCILLATORY))).
```

This should be seen as an "action definition" of (FLUCTUATING). That is, the abstract model knows that besides being viewed as a state, (FLUCTUATING) can also be viewed as

"repetitive change of an entity such that it both increases and decreases"[1]. Before being applied to the piece under consideration, the suggested focussing transformation must be checked against the TOWARD to see if it passes the restrictions. It does, since there is no more restrictive version of (FLUCTUATING) in the TOWARD. The transformation can therefore be applied to the piece under consideration.

Note that the use of an elaboration is not always a simple substitution. In this case, the focussing mechanism sees that (FLUCTUATING) was in the state configuration, and that the elaboration indicates an action. Furthermore, the NIL in the elaboration says that that part is unspecified, and should be substituted for when the elaboration is applied. Essentially, this means that "whatever was in the (FLUCTUATING) state before should now be acted on by (CHANGE)". Therefore, the piece under consideration becomes

```
(ACT-ON (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
        ((CHANGE)
            MODIFICATION (REPETITIVE OSCILLATORY))).
```

Comparing this with the input piece

```
(ACT-ON (PEOPLE (EMPLOYED (WORKING)))
        ((OR (HIRE) (LAYOFF))
            MODIFICATION (FREQUENT))),
```

we can see things are coming along. In fact, comparing the two pieces again, the connection sees that everything matches except

```
((CHANGE)
    MODIFICATION (REPETITIVE OSCILLATORY))
```

and

[1]Some of the more general concepts like (FLUCTUATING) have several elaborations which are appropriate in various circumstances. (FLUCTUATING) has an "effect" elaboration, an "action" elaboration, and a "curve description" elaboration.

((OR (HIRE) (LAYOFF))
MODIFICATION (FREQUENT)).

Therefore, the problem now is to decide whether "frequent hiring and layoff" is the needed "repetitive oscillatory change". Now, as discussed earlier, (CHANGE) is a generic which can be transformed into practically any action. (OR (HIRE) (LAYOFF)) is certainly a valid candidate for such a transformation. However, this does not imply that the transformation can really be applied. Indeed, it is necessary to show that (OR (HIRE) (LAYOFF)) is capable of changing (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL) before the transformation is allowed. Now "capable of changing" in this sentence is defined by the modelling knowledge attached to (CHANGE). This of course says nothing specific about (HIRE) or (LAYOFF). Instead, it knows about three ways in which entities can be changed: by being acted on by actions in a flow, by being DETERMINED-BY (or DETERMINED-FROM) a numerical function in which one or more of the determining factors change, or by having the thing it DEPENDS-ON change. Unfortunately, none of the above applies to (OR (HIRE) (LAYOFF)).

The program is now in a state in which it is missing information needed to make a connection. That is, it is not trying to apply abstracters or transformations; it is trying to see if a particular suggested transformation ((CHANGE) to (OR (HIRE) (LAYOFF))) is valid. As discussed in the previous chapter, PURPOSE's are used to supply the needed information in these cases. In particular, the inability of the program to apply the given modelling knowledge about ACT-ON's calls for a MODEL PURPOSE to model the flow containing (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL). In this way the program can see whether or not the deduction possibility given in the modelling knowledge may be used on the piece under consideration.

For (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL), this means modelling the workforce flow. Therefore,

```
(PURPOSE (MODEL
  (ACT-ON
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
      (CHANGE))))
```

is added as a constraint in the trend (to make sure Alfred knows why it is modelling the labor sector at this point; i.e, to find out what changes the workforce level), and the basic idea of the labor sector of the workforce-need theory (the chunk shown in Chapter I),

BASIC IDEA

```
(DETERMINED-BY
  (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
    (INFLUX (DELAY1 (ACT-ON (PEOPLE)
                            ((FIRM) (HIRE)))))
    (OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                             ((FIRM) (LAYOFF))))),
```

is made the chunk under consideration.

Since the **BASIC IDEA** chunk is at the appropriate LEVEL according to the trend (see p. 189)--although, since it's under a PURPOSE, it needn't be (see VII-3.2)--and since nothing else is violated, the chunk is set up as is and the association effort begins as before. Again, this large chunk has no abstracters, etc., so the connection process recurses down uneventfully until it gets to

```
(ACT-ON (PEOPLE) ((FIRM) (HIRE))).
```

Now, as discussed before in relation to DETERMINED-BY, the usual procedure for modelling this piece would be to immediately go into the hiring policy of the firm (i.e, in

order to find out in detail what the given entity is determined by). However, because the LEVEL of the trend contains "(HIRE)" and the PURPOSE does not override this with a lower LEVEL ("(CHANGE)" is in fact at a higher level of detail than (HIRE)), a more detailed version of the piece will not be set up--yet. At this point, then, the program need only make sure that the piece as it stands is a valid model for the firm, i.e, that nothing contradicts the notion that there is a (HIRE) of some sort and that it applies to (PEOPLE). As discussed in the previous chapter, this kind of thing is handled by a CONTRADICTION? PURPOSE. Accordingly,

(PURPOSE (CONTRADICTION? (ACT-ON (PEOPLE) (HIRE))))

is added to the trend, supervising a scan of the problem description to look for evidence against this piece. The scan first looks for the stated hiring policy (if any), and then looks for any characteristics of that policy that contradict the given piece (such as "no hiring for the next 5 years" (which isn't so rare nowadays)). In the case of Dominion, no such untoward difficulties arise, and the presence of the (HIRE) action in the hiring policy (figure 26, sentence 10), the fact that its object is (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) INCREASED) (also sentence 10), and the absence of any other information on hiring is sufficient to fulfill the CONTRADICTION? PURPOSE (i.e., that there is *no* contradiction).

This in turn means that the model piece under consideration is satisfactory, and it is incorporated into the modelled system with the following information made available to the setup mechanism:

EDGE: (PEOPLE (EMPLOYED))

TREND: (ABBREVIATION ((HIRE) (SLUFFED))).

(PEOPLE (EMPLOYED)) is the presumed result of (HIRE), and thus forms the aspect which that modelled chunk presents to the other chunks of the modelled system--the first edge of this modelling effort.

Note the introduction of the SLUFFing mechanism discussed earlier. The SLUFFED designation, you remember, implies that Freddy had the opportunity to model at a deeper level, but chose not to (that is, was prevented from doing so by one of its own mechanisms). Actually, since every firm is expected to have a hiring policy, Alfred would go ahead and presume that one exists even if it didn't find one. If there were no hiring policy given, Alfred would have recorded a hiring policy of simply (ACT-ON (PEOPLE) (HIRE)) and entered

(ABBREVIATION ((HIRE) (ASSUMED)))

in the trend.

I might add that the SLUFFED ABBREVIATION also implies that everything else at the level of the SLUFFED object should be similarly abbreviated until the modelling mechanism is told otherwise (or until the trend changes). This is not implied by the ASSUMED ABBREVIATION, since the use of the assumption might have caused an otherwise unwarranted ABBREVIATION to occur--which of course need have no bearing on other entities in the model (that is, the fact that the program was forced to assume a particular model for a particular entity does not imply that any other entity model at that level can not be modelled in full detail: lack of information about one entity does not imply lack of information about another).

At any rate, the current theme, trend, edge state is:

```
THEME=
     ((ACCOUNT-FOR (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                ((OR (HIRE) (LAYOFF))
                                   MODIFICATION (FREQUENT)))
                               AGENT (FIRM))
                              NIL))
      (ELIMINATE (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                             ((OR (HIRE) (LAYOFF))
                                MODIFICATION (FREQUENT)))
                            AGENT (FIRM))
                           NIL)))
```

```
TREND
   CONSTRAINTS
      TOWARD (CAUSE-OF
              ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                       ((OR (HIRE) (LAYOFF))
                          MODIFICATION (FREQUENT)))
                       AGENT (FIRM))
                    NIL)
      PURPOSE (MODEL
               (ACT-ON
                  (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
                  (CHANGE)))


   LEVEL
            FIRM-ACTION    (HIRE LAYOFF)
            FIRM-ENTITY    ((PEOPLE (EMPLOYED (WORKING))))


   ABBREVIATION ((HIRE) (SLUFFED)))


EDGE  (PEOPLE (EMPLOYED))
```

Next up in the **BASIC IDEA** chunk,

```
(DETERMINED-BY
  (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
    (INFLUX (DELAY1 (ACT-ON (PEOPLE)
                            ((FIRM) (HIRE)))))
    (OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                             ((FIRM) (LAYOFF))))),
```

is the DELAY1 piece on the (PEOPLE (EMPLOYED)) edge. As discussed earlier, DELAY is a generic which can be associated with a variety of problem description structures. If, on the other hand, the DELAY has a *particular* interpretation in a particular chunk, that is, if it is a place holder for one of a number of known possibilities, the possibilities can be

suggested by abstracters attached to it in that particular chunk (not to the DELAY generic itself). This is why any chunk or piece can have abstracters: to suggest particular matches for itself in particular circumstances (i.e., within a particular chunk). These abstracters are accessed and used by the connection mechanism just as it accesses and uses the abstracters attached to the entity itself (it looks for abstracters attached to the entity within a chunk first before it looks at the entity in general). The abstracters attached to DELAY1 in **BASIC IDEA** have a variety of problem description formulations to suggest for the DELAY, including (as a last resort) the ASSUMPTION that the DELAY does not exist in this problem. However, the abstracter that comes home in this case is the one which says that an INFLUX DELAY on (PEOPLE (EMPLOYED)) that yields (PEOPLE (EMPLOYED (WORKING))) may be (TRAIN). The connection mechanism then notes that DELAY1 is on the (PEOPLE (EMPLOYED)) edge, and, checking the rest of the **BASIC IDEA** chunk, it sees that indeed (PEOPLE (EMPLOYED (WORKING))) is on the receiving end of the "influx". It then initiates a search for some description of "training" in the problem description, and it soon finds one (sentence 23). This suggests the model

```
(ACT-ON (PEOPLE (EMPLOYED))
        (TRAIN))
```

for DELAY1. Since (PEOPLE (EMPLOYED)) is the result of (HIRE), and (HIRE) is in the LEVEL of the trend, and (PEOPLE (EMPLOYED)) is the object of the ACT-ON containing (TRAIN), (TRAIN) is considered to be at the right level of detail (see the rules in Chapter VII). The training model is therefore acceptable at that level of detail, and no other elaboration is set up. This leaves the current state of association of the chunk at

```
(INFLUX
    (ACT-ON (PEOPLE) (HIRE))
    (ACT-ON (PEOPLE (EMPLOYED)) (TRAIN))).
```

Note that the INFLUX entity now groups hiring and training. As discussed earlier, this implies that FLOW consistency checking has been done (since INFLUX is treated like FLOW for reasoning purposes) and that the flow so far is okay. The program will hereafter be able to use its knowledge of how flows work to reason about the relationship between these two actions; we'll see an example of this in a second.

The edge of the INFLUX part is taken to be (PEOPLE (EMPLOYED (WORKING))), the result of the (TRAIN) action of the FLOW (again, INFLUX behaves just like FLOW for reasoning purposes), and thus (via FLOW rule 5) the result of the whole FLOW. This edge is the one encountered by the

(OUTFLUX (DELAY2 (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
((FIRM) (LAYOFF)))))

part of the **BASIC IDEA** chunk as it is set up to be associated. In fact, this edge is quite important for this modelling effort, since it immediately indicates that DELAY2 should not be modelled as a separate action[1]. In general, this DELAY is just like the other one: it can take the place of a variety of model constructs. For example, it can represent a re-training delay in which people cease to become full-time workers, but are still employed. It can represent employees in transit from one firm location to another. Finally, it can represent a buffer between two different employee activities of the firm: the people cease to be "workers" when they are not working on one job or the other, but they are still employed (i.e., they haven't been laid off yet). However, in the modelling effort for this case, the edge indicates that none of these possiblities even have to be investigated. This is because it is given in the chunk that (LAYOFF) acts on (PEOPLE (EMPLOYED (WORKING)))--the very edge itself. That is, the situation looks like:

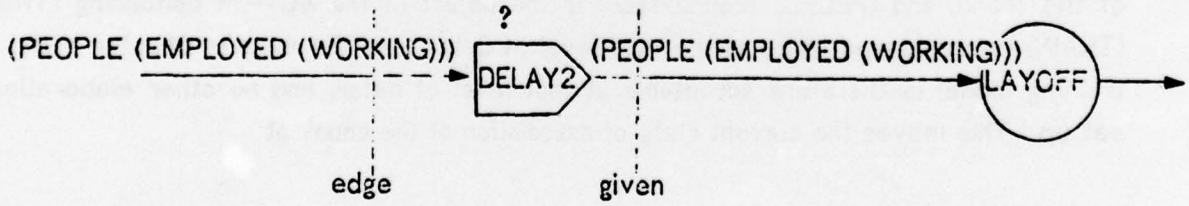


Figure 27. Use of the edge to edit the chunk

[1]Note that both INFLUX and OUTFLUX are modelled from the point of view of the DETERMINED-BY entity. Thus, for the INFLUX, the fact that (HIRE) is DELAYed means that the DELAY is between HIRE and the DETERMINED-BY. Being leftmost in the flow, (HIRE) is modelled first. For OUTFLUX, the same structure implies that the DELAY is leftmost, and thus must be modelled first as it will be on the edge which is currently *the* edge.

The program then checks to see if DELAY2 is a necessary characteristic of the workforce flow, which it is. Thus, there is a quandary: the edge dictates that DELAY2 should not be modelled explicitly, and the chunk dictates that some delay must be present. Of course, there is a way out of this difficulty, and the program knows about it: if the (LAYOFF) action itself can be modelled as a DELAY, everything is okay. Alfred therefore checks under (LAYOFF) to see if it has a DELAY model, which it does. DELAY2 is therefore edited out of the chunk (this is an example of model tailoring). The rest of the OUTFLUX chunk is okay, and it is set up as

```
(OUTFLUX
   (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
           ((FIRM) (LAYOFF))),
```

with a little tag on (LAYOFF) that indicates that it must model a delay. I went through this little detail of program operation to show that the edge really can affect the set up of chunks in a rather sophisticated way, and to show how the modelling effort leads the program to use its built-in knowledge. This knowledge is just sitting around in all of the "right places", waiting to be used. In this case, even though the obviously correct place to put the DELAY model of (LAYOFF) is under (LAYOFF), the program would never have found it by pursuing its then current line of investigation (it wasn't thinking about (LAYOFF) yet, only the delay it had to handle). Nonetheless, the modelling mechanism managed to call for the right piece of knowledge when it was needed--the chunk didn't have to jump out and declare itself.

At any rate, after this little involvement in setting up the chunk, its association is quite straightforward. Since (HIRE) was SLUFFED, and since we know from the LEVEL part of the trend that (HIRE) and (LAYOFF) are at the same level of detail, (LAYOFF) must also be SLUFFED. All that remains to be done is one of our little CONTRADICTION? checks like the one for (HIRE), this time with the added proviso that (LAYOFF) must model a delay. Happily, sentences 11 and 24 of the problem description provide the needed confirmation of the model, with the five day "notice period for layoff" being taken as the appropriate layoff delay (the DELAY model of (LAYOFF) was on the watch (via abstracters) for something like this). This results in the following finished association for the **BASIC IDEA** chunk:

WORKFORCE

```
(DETERMINED-BY
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
   (INFLUX
      (ACT-ON (PEOPLE)
              (HIRE))
      (ACT-ON (PEOPLE (EMPLOYED))
              (TRAIN)))
   (OUTFLUX (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                    (LAYOFF)))).
```

It looks like not much has happened to the original chunk given for **BASIC IDEA** on p. 197, and in some sense this is true: Fred has simply verified that the chunk is appropriate for this problem at a very general level (which is what was wanted). At any rate, this chunk is finished for now and we can move on to a more interesting interaction of the PURPOSE and TOWARD constraints in the trend. These constraints are, to refresh your memory,

```
TREND
   CONSTRAINTS
      TOWARD (CAUSE-OF
              ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                       ((OR (HIRE) (LAYOFF))
                          MODIFICATION  (FREQUENT)))
                  AGENT (FIRM))
               NIL)
      PURPOSE (MODEL
              (ACT-ON
                 (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
                 (CHANGE))).
```

Remember that the PURPOSE constraint in the trend requires that the modelling effort establish a connection between the recently completed **WORKFORCE** chunk and

```
(ACT-ON
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
      (CHANGE)).
```

This can be done almost immediately, now that Alfred knows the flow relationships around (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL). As I said earlier, the modelling knowledge of (CHANGE) considers that if an entity is acted upon by actions in a flow, those actions may model the desired (CHANGE). Since the **WORKFORCE** chunk models *all* (it is a DETERMINED-BY) of the flow actions which can affect (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL), the PURPOSE is considered to be satisfied. That is, **WORKFORCE** successfully satisfies

```
(ACT-ON
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
      (CHANGE)),
```

and the PURPOSE constraint, now fulfilled, can be "popped".

Let's quickly review the line of reasoning up to now in order to understand what's going to happen. The basic goal so far has been to see if the symptom under consideration (see p. 188) corresponds to the workforce need theory problem definition. Since initial attempts to establish the correspondence were unsuccessful, Alfred decided to take a slightly deeper look at the way workforce was determined in order to understand what was going on. At each stage (i.e, every time an association was incorporated into the modelled system), the TOWARD constraint has forced Freddy to ask "knowing what I know now, is there a connection between the symptom and the problem definition?" In the program, asking this question takes the form of suggesting focussing transformations for the chunk (see **VII**-3) and seeing if they can be successfully applied. With this in mind, let's see how the question is asked and answered.

In terms of chunks, the question becomes: "knowing that

```
(ACT-ON
     (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
     (CHANGE))
```

is modelled by

WORKFORCE

```
(DETERMINED-BY
     (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
  (INFLUX
     (ACT-ON (PEOPLE)
             (HIRE))
     (ACT-ON (PEOPLE (EMPLOYED))
             (TRAIN)))
  (OUTFLUX (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                   (LAYOFF))),
```

does

```
(ACT-ON (PEOPLE (EMPLOYED (WORKING)))
          ((OR (HIRE) (LAYOFF))
             MODIFICATION  (FREQUENT)))
```

match

```
(ACT-ON
     (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
     ((CHANGE)
        MODIFICATION  (REPETITIVE OSCILLATORY)))?"
```

That is, is frequent hiring or layoff the needed repetitive, oscillatory change?

Taking this question step by step, we can see that

(OR (HIRE) (LAYOFF))

is the needed (CHANGE) by looking at the recently associated **WORKFORCE** chunk (this was the crucial step we couldn't do before now) and knowing that an action which affects any part of a flow affects the result of the flow (see the FLOW rules in V-1.2), i.e., a change in (HIRE) implies a change in (PEOPLE (EMPLOYED (WORKING))). The next problem is to decide whether or not the described change is (REPETITIVE). I said earlier that the abstracters of (REPETITIVE) can match (FREQUENT), but I also said that they were sometimes restricted from doing so by the restriction mechanism of the TOWARD or PURPOSE. In this case, since there is no more restrictive version of (REPETITIVE) in the TOWARD, all of the abstracters of (REPETITIVE) can be used (remember the restriction rules given in VII-3.1), and (FREQUENT) is a valid match. As an aside, when wouldn't this be a valid match? If the TOWARD contained an elaboration of (REPETITIVE) such as (NIL (CHARACTERISTIC ((PERIOD) MODIFICATION NIL))) (the entity or action is periodic) or (NIL MODIFICATION (SEASONAL)) (the entity or action is seasonal), the (FREQUENT)-matching abstracter of (REPETITIVE) would be screened out because it is not present on either of these two elaborations. (Seasonal variations are hardly frequent; periodic ones may or may not be--at any rate it would be wrong to match (REPETITIVE) with (FREQUENT) in a close interaction group whose TOWARD indicated that it was modelling a seasonal or periodic effect.)

Now there is the question of whether or not (OR (HIRE) (LAYOFF)) is (OSCILLATORY). This is answered by an elaboration of (OSCILLATORY) which says than an action is possibly[1] oscillatory, i.e., (ACT-ON NIL ((CHANGE) MODIFICATION (OSCILLATORY POSSIBLE)), if the action is in the configuration (DETERMINED-BY NIL (INFLUX (CHANGE1)) (OUTFLUX (CHANGE2))) (where (CHANGE1) and (CHANGE2), like DELAY1 and DELAY2 stand for unique instances of the same generic). This simply says that an action can change an entity in an oscillatory manner if it affects both the INFLUX and the OUTFLUX of the flow

[1]The only meaning of "possibly" here is that though Alfred initially treats possible effects as certain, i.e., it ignores the POSSIBLE modification, it retains the modification so that if something goes wrong later (e.g., the program can't find a new trend) this assumption can be found and checked with the user.

which determines that entity. Note that (CHANGE) must be a compound action, i.e., an AND or OR construct, in order for this elaboration to have a chance of matching. This elaboration is suggested by the focussing mechanism. The restriction mechanism (remembering the rule about generics) limits matching activities for (CHANGE1) and (CHANGE2) to application to (OR (HIRE) (LAYOFF)) only, since the (OR (HIRE) (LAYOFF)) in the TOWARD is considered to be a more restrictive version of (CHANGE) after (CHANGE) was expanded into it above. At any rate, this means that (CHANGE1) and (CHANGE2) must be associated with (HIRE) and (LAYOFF) (or vice versa--the order is unknown at this point). This association is not direct. Again the program is in a situation in which it must decide on the validity of a proposed focussing transformation. It therefore uses a SHOW-CAUSE PURPOSE (which in turn uses the flow reasoning given in Chapter V) to show that (HIRE) is a valid match for (CHANGE1) because it is a member of an INFLUX which affects the target entity (PEOPLE (EMPLOYED (WORKING))) and "actions in a flow affect things downstream". (LAYOFF) is immediately seen to be a valid match. Therefore, (OR (HIRE) (LAYOFF)) *is the desired* REPETITIVE OSCILLATORY (CHANGE), and at long last,

```
(ACT-ON (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
        ((CHANGE)
           MODIFICATION (REPETITVE OSCILLATORY)))
```

is seen to match

```
((OR (HIRE) (LAYOFF))
   MODIFICATION (FREQUENT))).
```

But this does not complete the overall connection effort for the problem definition chunk

PROBLEM DEFINITION

```
(CAUSE-OF
  (ACT-ON
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
      ((CHANGE)
          MODIFICATION  (REPETITIVE OSCILLATORY)))
  ((FIRM-ACTION)
          MODIFICATION  (UNDESIRED
                        UNACCOUNTED-FOR))).
```

Alfred must show that this whole **PROBLEM DEFINITION** chunk matches the symptom originally taken from the theme,

```
(CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                   ((OR (HIRE) (LAYOFF))
                        MODIFICATION (FREQUENT)))
                  AGENT (FIRM))
          NIL).
```

This is a matter of matching ((FIRM-ACTION) MODIFICATION (UNDESIRED UNACCOUNTED-FOR)) with the unspecification "NIL" in the input. *(FIRM-ACTION) is a very simple generic* which is a place-holder for any action whose agent is the firm. The overall matching task now could be stated in English as "looking for any undesired and 'unaccounted-for' action of the firm which could possibly cause a change in the workforce". Furthermore, since the symptom is in the TOWARD, this is restricted to mean "any (due to the NIL) undesired unaccounted-for action of the firm which causes change in the workforce by frequent hiring or layoff". Since the generic (FIRM-ACTION) has no input constuct to be compared to, and since any focussing transformation on (FIRM-ACTION) is allowed because NIL is maximally non-restrictive, the program is again in the situation that it must see whether a suggested transformation (in this case, any firm action) is valid.

As usual, this means a PURPOSE. Since this is a CAUSE-OF construct, a SHOW-

CAUSE PURPOSE is set up to find which firm actions can cause a change in the workforce via hiring and layoff. In general this would mean checking through the workforce flow which was modelled earlier, using the FLOW rules as well as other cause-effect tracing knowledge attached to SHOW-CAUSE (see previous chapter), in order to find the desired cause. In this case, however, SHOW-CAUSE can use a much simpler bit of the deductive knowledge attached to it: "*x* is the cause of *y* because *x* actually is *y*". That is, a good candidate for the cause of changing workforce via hiring and layoff is hiring and layoff itself. This association is allowable as long as

```
((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
         ((OR (HIRE) (LAYOFF))
              MODIFICATION (FREQUENT)))
 AGENT (FIRM))
```

can be shown to match the piece under consideration,

```
((FIRM-ACTION)
     MODIFICATION (UNDESIRED UNACCOUNTED-FOR)).
```

This time the connection effort is led off (after it is discovered that the piece under consideration has no abstracters) with a structural transformation

```
(FIRM-ACTION)      to      (ACT-ON NIL
                                   NIL
                             AGENT (FIRM)).
```

The TOWARD restrictions force the NIL's to be matched to (PEOPLE (EMPLOYED (WORKING))) and (OR (HIRE) (LAYOFF)) respectively, which is of course completely consistent with the needs of the input piece. The only problem now is to show that (OR (HIRE) (LAYOFF)) is UNDESIRED and UNACCOUNTED-FOR (and that the fact that it is (FREQUENT) doesn't contradict this. This is accomplished by three separate CONTRADICTION? PURPOSE's. The first shows that there is no contradiction if (OR (HIRE) (LAYOFF)) is assumed to be UNDESIRED, because the action occurs (as can be seen from

the report of the initial symptom-finding investigation on p. 188) IN-SPITE-OF a given firm policy (which is always considered to be an unwanted occurrence). The next one determines that it is okay to consider (OR (HIRE) (LAYOFF)) UNACCOUNTED-FOR because no REASON is given for its IN-SPITE-OF occurrence. Finally, the third CONTRADICTION? PURPOSE can find no contradiction between (FREQUENT) and (UNDESIRED) or (UNACCOUNTED-FOR). Therefore, the match is allowed, and the problem definition of the workforce-need theory is seen to match the presenting symptom of the problem description in the Dominion case.

The successful completion of these associations means that the workforce need theory is indeed applicable to the problem at hand. All of this modelling work (i.e., everything in the last ten pages) was necessary to show this applicability. Indeed, a fairly extensive effort is expected at this stage because of the very abstract level of the theory's problem definition and the problem-specific nature of the symptom description. Since Freddy has been rather careful about selecting a theory, it can now be confident about devoting a large modelling effort to solving the problem in the case. That is, the initial problem-modelling work will serve the program in good stead; it has essentially set up a framework in which Alfred can view the rest of the problem-solving task. The remainder of the modelling effort will be much more detailed, but can be less subtle due to this earlier work.

Let us then proceed with the rest of the modelling of Dominion, but at a bit quicker pace. To begin with, the current theme, trend, edge state left by the theory selection effort is:

```
THEME=
     ((ACCOUNT-FOR (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                      ((OR (HIRE) (LAYOFF))
                                          MODIFICATION (FREQUENT)))
                                      AGENT (FIRM))
                              NIL))
     (ELIMINATE (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                      ((OR (HIRE) (LAYOFF))
                                          MODIFICATION (FREQUENT)))
                                      AGENT (FIRM))
                          NIL)))
```

```
TREND
  ABBREVIATION
          ((((FIRM) (HIRE)) (SLUFFED))
           (((FIRM) (LAYOFF)) (SLUFFED)))
EDGE
```

We see that the theme has not been altered (Freddy is still accounting for the effect). Most of the processing has been in terms of the trend and the edge (which is usual). In fact, the trend is about to change. With the completion of the **PROBLEM DEFINITION** and **BASIC IDEA** chunks, the fulfillment of the PURPOSE's within the trend, and the full use of the TOWARD constraint to restrict the modelling effort, the first trend terminates normally: there are no pieces to set up, and the TOWARD has been satisfied. Since the CONSTRAINT and LEVEL information of the trend has been fully utilized (and is in fact no longer useful), it is "popped", leaving only the ABBREVIATION's as reminders of unfinished business. There is no leftover edge, because there is no chunk actually hanging fire--they've all been associated and incorporated at some level of detail.

As discussed earlier, in this situation a new trend is set up with its TOWARD taken from the ABBREVIATION. The program therefore selects ((FIRM) (HIRE)) itself as the TOWARD of the new trend, since it is what is to be modelled, and since the basic idea of its containing sector (the workforce sector) has already been incorporated into the modelled system. This is simply a direction to the modelling effort to restrict its attention to (HIRE), given the framework provided by the already incorporated pieces of the workforce sector. There is no initial LEVEL information because there are no actions or entities contained in the object of the TOWARD ((HIRE) itself is not included). There is no edge yet because no actual modelling has been done. Alfred therefore sets up its model of the (HIRE) action, which is, as we saw earlier:

HIRE

```
(SMOOTH (DISCREPANCY
        ((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
            MODIFICATION (DESIRED))
        (NUMBER-OF (PEOPLE (EMPLOYED)) NIL))).
```

The association of this chunk is a fairly straightforward "expansion"-type process, so instead of giving a blow-by-blow account of what piece is set up when and what all changes are made to the trend and edge, I will simply give the results of the modelling effort and discuss a few of the more interesting issues. The model of **HIRE** finally determined by the association effort is:

```
      ((FIRM) (HIRE))
*1*         (RATIO
*2*           (DISCREPANCY
*3*              (SUM (RATIO (((CUSTOMER) (ORDER)) (SMOOTHED))
*4*                          (NUMBER-OF (LINE) 1000.))
*5*                   (RATIO
*6*                     (RATIO
*7*                        (DIFFERENCE
*8*                           (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
*9*                           ((NUMBER-OF (LINE) 1000000.)
                                 MODIFICATION (DESIRED))
*10*                       (NUMBER-OF (DAY) 10.))
*11*                    (NUMBER-OF (LINE) 1000.)))
*12*              (SUM (NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
*13*                   (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
*14*                   (NUMBER-OF (PEOPLE (LAID-OFF (WORKING))) NIL)))
*15*         (NUMBER-OF (DAY) 5.))).
```

Lines *3* - *11* are the model of the "desired personnel level", i.e., the

```
((NUMBER-OF (PEOPLE (EMPLOYED)) NIL)
    MODIFICATION (DESIRED))
```

piece of the **HIRE** chunk. *12* - *15* cover the "current personnel level" piece

```
(NUMBER-OF (PEOPLE (EMPLOYED)) NIL).
```

The DESIRED term consists of the number of people needed to handle the average customer order rate plus the number needed to bring the order backlog down to its

desired level of 1000000 LINE's. The abstract model chunks which supply the models for these elements are found in section V-1.3. The "current personnel level" term consists of the sum of three groups: people in training ("(EMPLOYED)"), people who have passed through training and are working ("(EMPLOYED (WORKING))"), and people who have been laid off but haven't left yet ("(LAID-OFF (WORKING))"). I'll talk about where these come from in a second.

The "10 days" which mysteriously appear in line *10* and the "5 days" in line *15* are the time spans over which backlogged orders and people (respectively) are "smoothed" in the sense discussed in V-1.1. That is, they are the averaging times which must be found in connection with the SMOOTH modelling entity. In order to find these times, the SMOOTH is treated as a special kind of first order delay, and the delay time is determined just as for any other DELAY. That is, the abstracters under DELAY which know how to look for the time constant associated with their subject look for some indication of the time needed to do the SMOOTH. If there is no explicit indication (which is pretty often) the manager is asked for an estimate. In the case of this **HIRE** chunk, however, all of the necessary information is given right in the case description. The 5 day figure for observing the discrepancy between desired and actual personnel comes from the program's summation of the total time needed for people to get through the input pipeline. Here the total is just the training delay given in sentence 23. 10 days is the time length of the industry's (and thus, in the absence of other information, Dominion's) average backlog, conveniently given in sentence 15 (Alfred knows that one work week equals five days). If the backlog were given in "LINE's", as one would expect, the program would have done the necessary "conversion"--a process we'll see later in this same effort. At any rate, this ten day figure is picked up as the needed smoothing constant for observing the backlog.

I'll say a bit more about a few of these associations, just to give you an idea of the level of effort involved.

For instance, finding (line *9*) that the "DESIRED" level of (ORDER (BACKLOGGED)) is 1000000 LINE's requires the following reasoning chain... We are originally told (see sentence 10 of figure 26) that "hiring" takes place when the order backlog is large. The program's only mechanism for modelling this kind of personnel need is via the "desired extra" workforce models shown in V-1.3. The appropriate elaboration, supplied by the focussing mechanism when it compares the piece under consideration to sentence 10, is:

```
(RATIO (SMOOTH (DIFFERENCE
        (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
        ((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
              MODIFICATION (DESIRED))))).
```

That is, the suggested model is that extra personnel are hired to reduce backlog (mechanism so far unknown). Since (ORDER (BACKLOGGED)) is readily associated with the customer order backlog described in sentence 10 by abstracters attached to (ORDER (BACKLOGGED)), the first real modelling question concerns the DESIRED term: what is the "desired" level of backlog in this firm? As in previous problems, Alfred is now in the position of having to check the validity of a proposed transformation--in this case, the use of the desired backlog term. This is done, as before, via a MODEL PURPOSE with the questionable piece,

```
((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
    MODIFICATION (DESIRED)),
```

as the PURPOSE constraint. Therefore, the program must model what "desired backlog" means in the Dominion case. Piece by piece comparison with sentence 10, suggests a standard DEPENDS-ON elaboration of (DESIRED),

```
(DEPENDS-ON (NIL1 MODIFICATION (DESIRED))
            (STATE-OF NIL1 NIL2))
```

(where NIL1 and NIL2 are uniquified NIL's--similar to DELAY1 and DELAY2, etc.), which yields the form

```
(DEPENDS-ON ((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                        MODIFICATION (DESIRED))
        (STATE-OF (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
                (LARGE))).
```

This transformation is allowable because the suggested (STATE-OF x) elaboration is more restrictive than the x form in the PURPOSE. The program must now determine how "large" the backlog can be before hiring should take place, since this is what defines the (LARGE) state in the above chunk. This means setting up another MODEL PURPOSE to investigate this matter. In order to satisfy this PURPOSE, Alfred looks at the order backlog information in the firm, e.g., sentence 31, but finds nothing helpful. It then checks the characteristics of the (PEOPLE (EMPLOYED)) in the firm (e.g., sentences 20, 25, etc.), but there is nothing that can help it decide what a "large enough" backlog state to require hiring is. Finally, it checks the unattached cause-effect information given in the problem description and finds something useful (sentence 14). This sentence relates a state of the backlog to a known "bad effect" (decreased sales). In the absence of other information, this means that the backlog state which is "not greater than" the given specification (in this case "large") is a desired maximum level, i.e., the quantity we are looking for. Thus, this PURPOSE is popped, and Alfred goes back to the original PURPOSE of modelling the desired backlog level as a dependency on a large state of the backlog. Now that the value of (LARGE) has been determined as a number ("(NUMBER-OF (WEEK) 2)"), the

(DEPENDS-ON (NUMBER-OF x NIL)
(STATE-OF (NUMBER-OF x NIL)
number))

structure can be converted (via a structural transformation[1]) into simply

number,

or, in this case, (NUMBER-OF (WEEK) 2), for use in the piece

[1]Actually, two transformations must be used: one which says that (STATE-OF (NUMBER-OF x NIL) *any specific number*) is equivalent to *that specific number*, and one which says that (DEPENDS-ON (NUMBER-OF x NIL) *any specific number*) is equivalent to *that specific number*.

```
(DIFFERENCE
      (NUMBER-OF (ORDER (BACKLOGGED)) NIL)
      ((NUMBER-OF (ORDER (BACKLOGGED)) NIL)
            MODIFICATION (DESIRED))),
```

(which is necessary because DIFFERENCE insists on working on (NUMBER-OF...) constructs only). Furthermore, in associating the DIFFERENCE chunk, the modelling information attached to DIFFERENCE which originally makes sure that both specifications are (NUMBER-OF ...) also checks the units the numbers are being expressed in: they must be the same for both terms. In this case, they are not, because order backlog is defined in terms of "lines" in this problem description (see, for example, sentence 31 of figure 26 which gives the current value of the Dominion backlog), not the "weeks" given in the newly discovered value for (LARGE). Therefore, Alfred has to set up yet another MODEL PURPOSE in order to convert the "2 weeks" into "lines".

This essentially comes down to the question (again "asked" in terms of chunk transformations) of how much output 2 weeks of production represents. That is, Alfred makes an attempt to convert a "2 week" chunk into an "x LINE's per 2 week" chunk. This involves an initial look for direct information, which fails, followed by conversion of "weeks" to "days" and another look at the characteristics of the firm. This time, sentence 29 is discovered, the necessary multiplication is done, and the PURPOSE is fulfilled. Alfred thus establishes the DESIRED level of (ORDER (BACKLOGGED)) for ((FIRM) (HIRE)) (it may be different for other things) at 1000000 LINE's, the term that appears in line *9*.

Another interesting aspect of this association effort is that Alfred must ask the user at one point whether or not "LINE's" can be inventoried (to see whether it should worry about reducing personnel needs when an inventory exists). Alfred has no a priori knowledge of "LINE's" and no way to figure out whether they can be stored (indeed, the first time it ever saw the word "line" was in this problem description). Thus, the problems of dealing with this unknown entity force interaction with the user. In real life, of course, the world knowledge of the consultant prevents such silly questions.

Other associations for **HIRE** are quite direct. For example, (PRODUCTIVITY)

(of individual employees--a key variable for the workforce need theory) turns out to require only the (CONSTANT) elaboration (see V-1.3) in this case; it is given to be "1000 LINE's" in the problem description statement about typesetters (sentence 22). Also, although the personnel breakdown on lines *12* - *14* may look complex, it is really easily derived. As I have said previously, before a FLOW entity is incorporated, certain consistency checks are made on it. In particular, Alfred automatically keeps track of entities after each ACT-ON of a FLOW (since ACT-ON'S change the state of the entity), in order to make sure that the FLOW conservation rules given in Chapter V are complied with in each flow of the model. Therefore, when the workforce flow was modelled previously during the theory selection trend, the check was made. In this case, the entities of the flow were (PEOPLE), the source; (PEOPLE (EMPLOYED)), after hiring; (PEOPLE (EMPLOYED (WORKING))), after training; (PEOPLE (LAID-OFF (WORKING))), after the layoff delay (remember that (LAYOFF) has a DELAY model in this case); and then (PEOPLE) again after the entire layoff activity. This set of entities complies with FLOW rule number 4, and is therefore allowable.

This checking of flow entities has another important use. As I have said many times before, actions and entities in Alfred's abstract model must be tailored to fit the needs of specific problems. For example, the interpretation of the general entity (PEOPLE (EMPLOYED)) may vary widely depending on the problem. It may be okay to consider a single (PEOPLE (EMPLOYED)) group for the model, or it may be necessary to consider several separately modelled groups all as (PEOPLE (EMPLOYED)) (e.g., if the firm has several different kinds of employees). Therefore, the exact structure of (PEOPLE (EMPLOYED)) depends on the close interaction group in which it is being considered. Whenever an entity is used, as (PEOPLE (EMPLOYED)) is in this (HIRE) case, Alfred always goes back to the basic definition of the entity and intereprets it in the current close interaction group (by looking at the TOWARD for that group, as discussed earlier). By "basic definition", I mean the facts about an entity (or action) that never change: the properties that come with it (see Table 1 of Chapter III). In the case of (PEOPLE (EMPLOYED)), it is seen that the action which puts (PEOPLE) into the (EMPLOYED) state is (HIRE), and the one which takes (PEOPLE) out of the (EMPLOYED) state is (LAYOFF) (or (FIRE)). Therefore, *in any problem* which Alfred can handle, (PEOPLE (EMPLOYED)) is always defined to be "what is just after (HIRE) to what is just after (LAYOFF)". By again using the FLOW rules on the previously modelled TOWARD for this close interaction group, Alfred sees that in the workforce flow for Dominion, this is the sum of the three entities named

above, i.e., lines *12* - *14*. This definition of entities in the context of their close interaction group is one of the basic model tailoring mechanisms of the program (the other is the "edge lapse" mechanism seen earlier).

This is all I have to say about the association of **HIRE**. Furthermore, since the modelling of the remaining ((FIRM) (LAYOFF)) ABBREVIATION is quite similar to that of ((FIRM) (HIRE)), I won't go into it at all here. Instead I'll just give the state of the trend and edge (the theme is the same as before) and move right along:

TREND

EDGE ((((CUSTOMER) (ORDER)) (SMOOTHED)) (ORDER (BACKLOGGED)))

The previous trends (the ones dealing with (HIRE) and (LAYOFF)) having been successfully completed, the current trend is empty. For a change, the edge is the most interesting feature. In particular, when (((CUSTOMER) (ORDER)) (SMOOTHED)) and (ORDER (BACKLOGGED)) were introduced into the association of **HIRE** (lines *3* and *8*, respectively), they became hanging edges (since they belong to unassociated chunks). These unfinished edges cause notations to be left in the trend, since, as discussed in the previous chapter, "funny" edges may indicate that a trend change is in the offing. Furthermore, since the trend has been closed with these edges still left hanging, and since they are both parts of other sectors, they must be entered onto the OPEN property of the theme. Thus, the new theme is:

```
THEME = ((ACCOUNT-FOR (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                          ((OR (HIRE) (LAYOFF))
                                              MODIFICATION (FREQUENT)))
                                          AGENT (FIRM))
                                 NIL))
         (ELIMINATE (CAUSE-OF ((ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                                       ((OR (HIRE) (LAYOFF))
                                           MODIFICATION (FREQUENT)))
                                       AGENT (FIRM))
                              NIL)))

   OPEN ((((CUSTOMER) (ORDER)) (SMOOTHED))
         (ORDER (BACKLOGGED))).
```

What this really means is that although the whole association effort so far has been completed to the desired level of detail, some loose ends have been left around. Since Alfred is still under an ACCOUNT-FOR theme, it will have to "close" these OPEN entities (everything must be closed in order to satisfy ACCOUNT-FOR) by modelling them at the highest level of detail it can get away with.

Accordingly, the basic idea of the customer order sector,

CUSTOMER ORDER

```
(SMOOTH (DETERMINED-FROM ((CUSTOMER) (ORDER))
                         (PRICE-EFFECT)
                         (DELIVERY-TIME)
                         (SELLING-EFFORT)
                         (QUALITY-EFFORT))),
```

becomes the TOWARD of a new trend, and Alfred sets about to model it. Note that the

"SMOOTHED" requirement of the OPEN specification is carried along to the trend as a specific request to model a SMOOTH of the quantity, not just the quantity itself. Again, this will be used by the modelling mechanism to suggest the appropriate focussing transformations on the chunk. Remember that DETERMINED-FROM simply gives the program a group of factors and tells it to model them before worrying about the thing that is actually "determined". However, just as was the case for **SYMPTOM**, this chunk has the proviso (in terms of attached abstracters) that the modelling effort should first check to see if anything is specifically said about customer ordering itself. In the case of Dominion, the abstracters pay off: sentence 4 says that customers consider both price and delivery time in making their orders. This is extremely useful in limiting Alfred's inquiry to only the first two factors of **CUSTOMER ORDER**[1]; however, it eventually turns out to be misleading. This is because Dominion's price is not significantly different from its competitor's prices. Alfred discovers this (it's in sentence 35) while it is trying to assess the (PRICE-EFFECT). Since the only model for (PRICE-EFFECT) models a function based on the discrepancy between firm and competitor prices, this means that there is essentially *no* (PRICE-EFFECT), so that the program must stop this association effort and change trends via the abnormal termination mechanism (there are no chunks to set up and the TOWARD has clearly not been utilized). As discussed in Chapter VII, this means that another elaboration of the basic idea of the customer order sector is selected for the TOWARD, and the trend is started. The program knows that the new elaboration should not include (PRICE-EFFECT). This leaves only the simpler elaboration of ((CUSTOMER) (ORDER)) I have singled out in Chapter V:

DELIVERY TIME EFFECT

```
(SMOOTH (DECREASING-FUNCTION
                (RATIO (ORDER (BACKLOGGED))
                       (PRODUCTION)))).
```

DECREASING-FUNCTION is another one of those special modelling entities like

[1] If the manager takes the trouble to specifically mention some components of the DETERMINED-FROM, it is assumed that *only* these components are of interest.

DELAY and SMOOTH which has its own set of rules and abstracters. It will allow any monotonically decreasing function of a kind that it knows about: decreasing exponential, decreasing linear, and a table of values in which each value is less than the previous one. The chunk is therefore set up as

```
(RATIO (ORDER (BACKLOGGED))
       (PRODUCTION));
```

the "DECREASING-FUNCTION" specification in the TOWARD will insure that it ends up being the right kind of function by restricting the abstracters to look for (or ask for) only acceptable decreasing functions. Actually, in most cases, including this one, as we will see, no function at all is given by the manager, and a default function must be supplied by the program (subject to the manager's approval, of course).

The first piece of the chunk that is set up is (ORDER (BACKLOGGED)), but since this entity has already been modelled as Dominion's customer order backlog, it is immediately incorporated into the modelled system (simply as (ORDER (BACKLOGGED))). This leaves the program the problem of modelling (PRODUCTION), which is a whole sector in itself.

Remember though, that since this is a trend started by an OPEN specification, the program will try to get away with the highest level of detail possible. Thus, shoving

```
(PURPOSE (MODEL (PRODUCTION)))
```

onto the trend, it sets up the simplest elaboration (least number of entities) of (PRODUCTION) it knows about:

```
(DETERMINED-BY (ORDER-FILLING)
    (TIMES (PRODUCTIVITY)
           (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL))).
```

This can be salted away immediately: both (PRODUCTIVITY) and (NUMBER-OF (PEOPLE

(EMPLOYED (WORKING))) NIL) have already been modelled. Thus, the PURPOSE is fulfilled immediately, with the

```
(RATIO (ORDER (BACKLOGGED))
       (PRODUCTION))
```

chunk modelled as

```
(RATIO
  (ORDER (BACKLOGGED))
  (TIMES
    (NUMBER-OF (LINE) 1000.)
    (SUM
      (NUMBER-OF (PEOPLE (EMPLOYED (WORKING))) NIL)
      (NUMBER-OF (PEOPLE (LAID-OFF (WORKING))) NIL)))).
```

Next comes the matter of the DECREASING-FUNCTION. After the abstracters search through the problem description for some notion of how the delivery delay affects customer ordering policy and find nothing, Alfred hauls out its default DELIVERY-TIME-FUNCTION and presents it to the manager for approval (I consider this better than just asking the manager to give some numbers). The function looks like this:

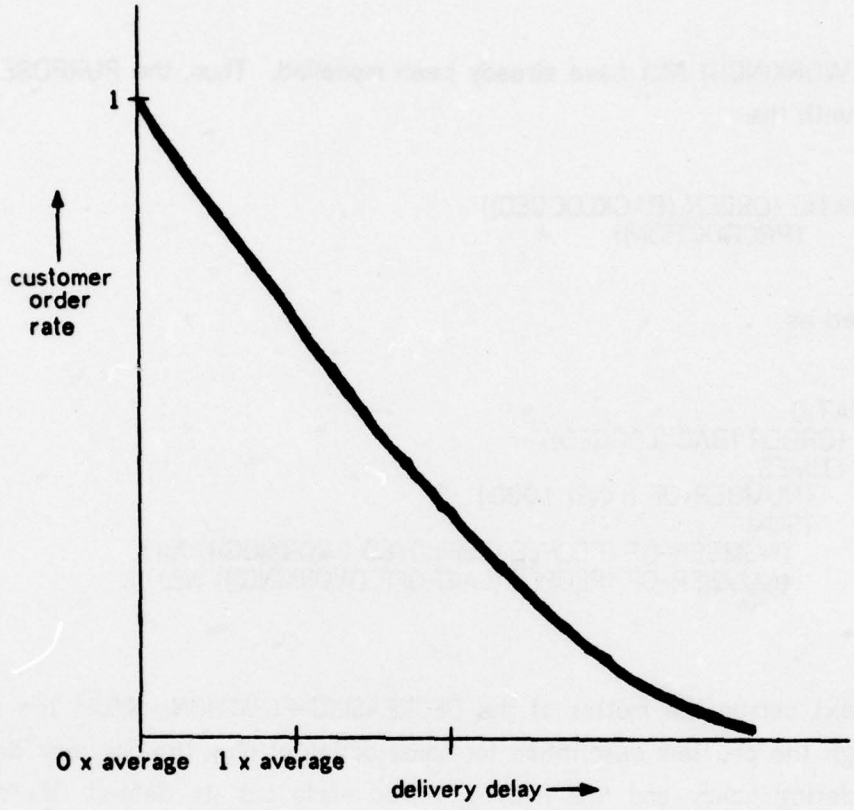


Figure 28. Alfred's DELIVERY-TIME-EFFECT-FUNCTION

Let us assume that the manager likes it (if he doesn't, he can give his own). Since this function was suggested by one of the abstracters attached to DECREASING-FUNCTION, it clearly satisfies the functional restrictions. This allows the program to set up

```
(TIMES
      (DELIVERY-TIME-EFFECT-FUNCTION
            (RATIO
              (ORDER (BACKLOGGED))
              (TIMES
                (NUMBER-OF (LINE) 1000.)
                (SUM
                  (NUMBER-OF
                    (PEOPLE (EMPLOYED (WORKING)))
                    NIL)
                  (NUMBER-OF
                    (PEOPLE (LAID-OFF (WORKING)))
                    NIL)))))
      (INPUT-ORDER-RATE))
```

as the current chunk. That is, DELIVERY-TIME-EFFECT-FUNCTION just gives a variable coefficient, as it were, that must be multiplied by the "raw" order rate to actually model customer ordering. All of this information (i.e., the stucture of this chunk) comes along with DELIVERY-TIME-EFFECT-FUNCTION. Once it is known that this function is going to be used, this chunk is immediately set up (other functions may behave differently).

Clearly, (INPUT-ORDER-RATE) is the next piece to be modelled. Actually, the abstracters attached to this piece are very similar in purpose to those of DECREASING-FUNCTION. They scan the problem description for information about customer ordering, and are ready to suggest default models if no information is forthcoming. Dominion is sort of half-and-half in this respect. Sentence 30 gives the *average* daily order rate, and nothing else. There is no other information baout order rates in the problem description. Actually, this is a rather common situation--managers tend to speak in terms of average quantities--and (INPUT-ORDER-RATE) has an elaboration ready to deal with it. Specifically, it suggests that a noise function be added to the given average rate to produce a realistic model of input behavior. Assuming that the manager is satisfied and supplies the parameters the noise function needs, the resulting association is

```
(TIMES
        (DELIVERY-TIME-EFFECT-FUNCTION
              (RATIO
                 (ORDER (BACKLOGGED))
                 (TIMES
                   (NUMBER-OF (LINE) 1000.)
                   (SUM
                      (NUMBER-OF
                        (PEOPLE (EMPLOYED (WORKING)))
                        NIL)
                      (NUMBER-OF
                        (PEOPLE (LAID-OFF (WORKING)))
                        NIL)))))
        (SUM (NUMBER-OF (LINE) 100000.)
              (NUMBER-OF (LINE) (NOISE
                 100000. (.01 DAY) (.01 MONTH))))).
```

This leaves only the SMOOTH to take care of to complete the requirements of the TOWARD.

As in the previous cases, this means that the SMOOTH-modelling abstracters take over. The form of the result is that of any other SMOOTH,

$$\text{smoothed quantity} = \frac{\text{quantity}_t - \text{quantity}_{t-1}}{\text{averaging time}},$$

or, in this case,

```
(((CUSTOMER) (ORDER)) (SMOOTHED))
  (RATIO
      (DIFFERENCE
        (TIMES
            (DELIVERY-TIME-EFFECT-FUNCTION
                  (RATIO
                    (ORDER (BACKLOGGED))
                    (TIMES
                      (NUMBER-OF (LINE) 1000.)
                      (SUM
                        (NUMBER-OF
                          (PEOPLE (EMPLOYED (WORKING)))
                          NIL)
                        (NUMBER-OF
                          (PEOPLE (LAID-OFF (WORKING)))
                          NIL)))))
            (SUM (NUMBER-OF (LINE) 100000.)
                  (NUMBER-OF (LINE) (NOISE
                    100000. (.01 DAY) (.01 MONTH)))))
        ((CUSTOMER) (ORDER)))
      (NUMBER-OF (DAY) 10.)).
```

The ten day averaging time is the same one used for smoothing orders in **HIRE**: the program tries to use the same averaging for the same quantity where possible. This, then is the final model for the **CUSTOMER ORDER** sector. Since the TOWARD (which was simply the basic idea of this sector) is satisfied, and there are no chunks left to set up, the trend terminates normally.

Alfred next concerns itself with the (ORDER (BACKLOGGED)) entity in the OPEN specification. The modelling of backlog is part of the **PRODUCTION** sector--the main part of which has already been modelled at a general level of detail. There is only one elaboration for backlog (see V-1.3). It is in fact a *definition*: the accumulation resulting from more orders flowing in than filled orders flowing out, or

```
(DETERMINED-BY (ORDER (BACKLOGGED))
   (INFLUX (ACT-ON (ORDER)
                   ((CUSTOMER) (ORDER))))
   (OUTFLUX (ACT-ON (ORDER (BACKLOGGED))
                    (PRODUCTION)))).
```

Since the two actions ((CUSTOMER) (ORDER)) and (PRODUCTION) have already been modelled, the association is immediate:

```
(DETERMINED-BY (ORDER (BACKLOGGED))
      (INFLUX
         (TIMES
           (DELIVERY-TIME-EFFECT-FUNCTION
                 (RATIO
                   (ORDER (BACKLOGGED))
                   (TIMES
                     (NUMBER-OF (LINE) 1000.)
                     (SUM
                       (NUMBER-OF
                         (PEOPLE (EMPLOYED (WORKING)))
                         NIL)
                       (NUMBER-OF
                         (PEOPLE (LAID-OFF (WORKING)))
                         NIL)))))
           (SUM (NUMBER-OF (LINE) 100000.)
                (NUMBER-OF (LINE) (NOISE
                  100000. (.01 DAY) (.01 MONTH))))))
      (OUTFLUX
         (TIMES
           (NUMBER-OF (LINE) 1000.)
           (SUM
             (NUMBER-OF
                 (PEOPLE (EMPLOYED (WORKING)))
                 NIL)
             (NUMBER-OF
                 (PEOPLE (LAID-OFF (WORKING)))
                 NIL))))).
```

With the completion of this chunk, the trend is again completely empty and, this time, there are no loose edges. This is an appropriate end condition for the ACCOUNT-FOR theme (the model of the thing to be accounted for is considered to be finished), which, having terminated normally, is removed from the theme, leaving Alfred with a complete modelled system and an ELIMINATE theme:

```
(ELIMINATE (CAUSE-OF
                (ACT-ON (PEOPLE (EMPLOYED (WORKING)))
                        ((OR (HIRE) (LAYOFF))
                        MODIFICATION  (FREQUENT)))))
```

to be considered. This condition is tantamount to a request to the system to "solve" the problem it has just found, in this case by eliminating a bad effect (the need for frequent hiring and layoff).

Alfred can now investigate this bad effect in terms of the modelled system--a piece of structure which it obviously knows a great deal about. In particular it now has two strings to its bow: it has the problem situation modelled, and it knows that the workforce need theory is applicable. As mentioned earlier, the workforce-need theory carries with it knowledge of how to try and fix what it has found to be wrong. In fact, it knows about a number of things which may be used to ELIMINATE various bad workforce-need effects (much as a doctor has a group of prescriptions and other procedures for dealing with a disease at various levels of severity). Alfred's approach is to first find the exact cause of the workforce need problem (there are several possibilities), and then to find a policy suggestion or some other recommendation which ameliorates that problem. This is the "diagnosis" phase discussed earlier. "Policy suggestions" are alternate elaborations of firm policies which satisfy the solution criteria set up by the diagnostic process. The other kinds of recommendations take the form of alterations to existing firm policies or characteristics. The diagnostic process for the Dominion case proceeds as follows:

The simplest cause of workforce fluctuation known to the theory is that the manager has simply neglected the inevitable delays of handling personnel--time lags in training, hiring, firing, etc. Alfred therefore checks the personnel flow to make sure that each action has a "reasonable" model, i.e., that some delay at least is shown. In this case, everything seems to be okay: training and layoff delays have been explicitly modelled, and

there is no reason to suspect that a hiring delay is necessary. This "possible cause" of the fluctuation is therefore abandoned. As discussed in Chapter V, the program knows about two other (closely related) causes. One is that although the delays themselves are taken into account by the firm, their dynamic effects on the personnel flow are mishandled, thus causing "out of synch" hiring and firing. The other is that although firm policies handle delays correctly all around, they are based on apparent workforce needs which neglect any natural workload-altering factors in the firm's environment (which are independent of the personnel policies). This problem gives rise to out of synch hiring and firing which is very similar to that caused by the other problem.

In fact, the outward signs of these two problems are virtually indistinguishable; there is no straightforward way for the program to tell them apart. Alfred therefore resorts to an "expert trick". It looks through the problem description for natural workload-reducing or -enhancing factors; if it finds none, only the delay-based cause is possible. If Alfred does find a workload-altering factor, it temporarily takes it out of the modelled system and redoes the analysis. If the policies are still at fault, the delay-based cause is again taken to be the culprit. However, if the policies now seem reasonable, the "phantom factors"-based cause must be at the root of the problem.

Alfred knows about a number of these natural workload-altering effects, ranging from depreciation of inventory, returned product handling due to poor quality control, etc. (workload-enhancing) to seasonal adjustments, advertising saturation, etc. (workload-reducing). For Dominion, there is only one such factor, the effect of delivery delay on customer orders (increased backlog causes increased delivery delay, thus reducing customer orders, thus naturally reducing backlog without the need of extra personnel). Alfred finds this factor and "turns it off" by setting the multiplicative delivery delay coefficient to 1. Analysis of the modelled system now shows that the policies are behaving okay--neglect of the delivery delay must be the cause of Dominion's workforce woes.

With the cause now found, the "prescription" facility is called in. I will just summarize the chain of reasoning that is carried out under the control of the various PURPOSE's and TOWARD's (each step corresponds to a change of constraint in the trend):

<> A workload-reducing factor is being ignored; therefore the firm should change its personnel policies to reflect an appropriately higher apparent workload.

<> The factor being ignored is the effect of the delivery delay on the customer's ordering policy--a factor which is beyond the control of the firm.

<> Therefore, find out what the delivery delay effect depends on.

<> Delivery delay depends on order backlog, which is controllable by the firm. The higher the backlog, the fewer the customer orders. Therefore, the higher the backlog, the greater the workload-reducing effect. Personnel policies should reflect this factor.

<> Are there any personnel policies (i.e., actions in the personnel flow) which already take backlog into account? Yes--(HIRE).

<> Does the (HIRE) action appropriately reflect a higher workload-reducing effect for higher levels of backlog? No--it uses a constant DESIRED backlog level to determine workforce needs.

<> Are there any elaborations of ((FIRM) (HIRE)) in the abstract model which correctly model the desired relationship between backlog and workload? No.

<> Are there any other ways of determining a desired backlog level which happen to have the right characteristics? Yes:

```
((ORDER (BACKLOGGED))
     MODIFICATION (DESIRED))
  (TIMES (INCREASING-FUNCTION (ORDER (BACKLOGGED)))
         (ORDER (BACKLOGGED))).
```

Alfred has now decided that the way to solve the problem is to figure out "desired backlog" on a sliding scale rather than using a constant value. That is, enhance the *apparent* desired backlog level in order to reduce hiring needs. Thus, if the backlog gets up to 4 weeks, 3 weeks, say, rather than 2 should be the apparent desired level in order to correct for the natural fall off in customer orders.

As with all of the other FUNCTION's, the program will suggest a default INCREASING-FUNCTION (a simple ramp function, since it doesn't know anything about the desired characteristics of the function) or get input from the manager[1]. Assuming that some function is associated with INCREASING-FUNCTION, Alfred's activities on the Dominion case are complete.

[1]It is quite possible for the manager to improve on Alfred's suggested solution by providing an INCREASING-FUNCTION which is tuned to the particular DELIVERY-TIME-EFFECT in the environment. Alfred isn't quite up to this by itself. However, it turns out that for the actual range of fluctuation likely to be encountered in Dominion's backlog, any "reasonable" function is perfectly adequate.

Chapter IX
Conclusions

Having now examined the implementation methodology in some detail and shown an example of its use in expert problem-solving, I think that it is possible to discuss its implications on the building of future expert systems and its limitations as a technique for writing knowledge-based programs. Because there is not yet any science of expert systems, nor even any agreed upon set of benchmarks, this discussion will necessary be comparative (harking back to the spirit of Chapters II and VI), and somewhat speculative. Nonetheless, I think that one of the major contributions of a thesis like this is that it provides a point for comparison in thinking about future efforts. An expert system builder should be able to learn something about the applicability of a proposed or existing approach by seeing how its underlying assumptions compare with the assumptions needed to make the focussing method of this thesis work. He should also be able to judge the effectiveness of the approach by comparing its solutions to common problems with the solutions proposed here. Of course, this comparison is only possible if it is quite clear what this thesis stands for. I will use this chapter to try to clarify what assumptions, tacit and otherwise, have been made in order to make the focussing approach operable, what solutions I have found to problems that are liable to crop up again, and what the limitations of these solutions are.

Let me start by stating briefly what I think this thesis offers for the future. First of all, it must be emphasized that the implementation methodology proposed here is fundamentally a mapping engine (which is why I often call it the focussing methodology even though, strictly speaking, that term applies only to the connection process of the program). That is, because of the nature of reformulation, which says that the process of constructing the mapping between the expert domian and the input domain is the "hard part" in the type of problem-solving it is used for, almost all of the interesting techniques developed in this thesis have to do with this map construction (or "connection", or "pattern matching") process. The problem of knowledgeable pattern matching, the use of knowledge base semantics to be smarter about doing pattern matching, is of much current interest in the field of building expert systems, and I think that this thesis offers one consistent

approach to the problem. In particular, I think that there is a very useful notion of controlling the connection process by looking at the previous connections that have been made--using what has been learned about how the connection process has been successfully applied in the past to restrict the way it should be applied in the future. Along with this there is the ancillary point that this "feedback" information from the previous connections should consist of selected features of those connections and that these features should be maintained in a special context mechanism. Also, the technique of doing the connection by applying transformations to the pieces in the expert knowledge base rather than the pieces that come in as input, though very closely allied to the above feedback notion, is worth pointing out separately because it is seen rather rarely in existing expert systems. Finally, I think that the idea of using the system's connection mechanism to control the level of detail of the modelling effort and the selection of relevant information in the problem description is important for making expert systems that work in reasonably large domains. Existing systems pay almost no attention to these problems, primarily, I think, because their connection mechanisms are not flexible enough to handle them. The focussing scheme at least raises the possibility of addressing these issues and provides a first order solution to them in a limited class of domains.

I think that the contribution of this thesis is that it describes a specific implementation methodology for putting this feedback-controlled focussing kind of connection into a computer program. By this I mean the definition of the theme, trend, edge, mechanism in terms of the features that should be selected for feedback and directions for the use of these features; the breakdown of the connection process into the use of a specific set of structural transformations and elaborations; the defintion of the role of canonicalization; etc. The benefit comes from the fact that the particular definitions used for these concepts in Alfred can be examined to see where they can be extended and where they are limited by fundamental characteristics of the design. I will discuss both of these issues below.

The other important notion in this thesis is that pieces of the expert model can be put into the system in a fairly independent way (i.e., without having to plan for all of the uses that the system might have for the knowledge contained in them), without sacrificing control of the problem-solving mechanism to an exhaustive search mechanism or waiting for parallel schemes to become feasible. There are certainly limitations on the domains to which the techniques of this thesis can be applied (and these will also be

explored below), but I think that it is very useful to see one of the possible ways in which the characteristics of the problem-solving task can be used to make this much needed chunk independence feature implementable in expert systems with current technology.

Now that I have defined the basic areas in which this thesis may prove useful, we can go on to a discussion of the more detailed issues of where the limitations of the focussing methodology lie, and why they arise.

There are two ways of looking at the issue of limitations. One is to examine the focussing methodology from the point of view of the reformulation approach to problem-solving to see which of the assumptions built into focussing come directly from the notion of reformulation. The builder of a proposed system can then find parts of the problem-solving method of his system which share assumptions with the reformulation approach in order to see whether Alfred-like techniques can be applied to those parts. The other way is to look at the focussing methodology from the point of view of "services" that an expert system builder might want. That is, the system builder can see which of the features he wants in an implementation methodology are best provided by focussing and which are best provided by other methods. He can then make design decisions based on trade-offs between the various methodologies available. I will therefore begin with a discussion of how the assumptions of the reformulation approach influence the focussing methodology and limit its applicability, and then go on to a comparison of the services offered by focussing with those offered by other techniques in the current sample space of expert systems.

I identify the following assumptions, basic to the focussing methodology, as coming directly from the reformulation approach to problem-solving:

(1) There are relatively few concepts in the abstract model, and they are canonical and well defined.

(2) There are relatively many input concepts.

(3) The map construction phase of the problem-solving sets up a reasoning environment in terms of which the problem is solved.

(4) The deductive part of the abstract model consists of relatively few general procedures which work in terms of abstract concepts, but cannot work on input concepts.

(5) The problem can be solved (i.e., any analysis and debugging that is necessary can be done) readily once the appropriate mapping between the expert model and the input has been set up.

(6) The matching tricks, i.e., the direct correspondences between abstract concepts and input concepts, in the expert model are poorly organized and are not uniformly distributed over the domain. In particular, there is no complete, direct mapping between structures of abstract concepts and the various structures of input concepts they can represent.

Let's take these assumptions one at a time and see how they affect the design of the focussing process and what limitations they present.

I have said several times before that the combination of assumptions (1) and (2) is the basic motivation for the focussing approach. That is, since the concepts of the abstract model are fewer and more well known, it is reasonable to think about applying a series of match-directed transformations to them rather than to the input. Clearly, in any domain in which this situation was reversed, focussing as I have defined it here would be a poor choice. It might be argued that such a situation would never come up, since expert models, even if they are not of the compact abstract model variety used in reformulation, always act on *classes* of input problems, and thus must be more aggregate than the input domain. Indeed, it is hard to find an exception to this. However, there are consulting methods (notably medical diagnosis--see [Miller]) in which the expert severely limits the possible input by careful (and rigid) questioning. If this forcing of desired input in this way is allowed, it is quite possible that the situation in assumptions (1) and (2) would be changed to the degree that it would be better to concentrate on input-driven methods.

Furthermore, there is a great deal more buried in the first two assumptions than the relatively straightforward point above. As I have also tried to emphasize in the body of the thesis, the restriction control structure, in which only appropriate matching alternatives are allowed to be used at any given point in the modelling effort, prevents the

combinatorial explosion problems that arise in other techniques (e.g., unrestricted production rules and pattern-directed invocation). Therefore, simply raising the number of concepts in the abstract model does not pose any serious limitation on the focussing method. However, note carefully that focussing, in particular the restriction technique, relies on assumption (1) being literally true: the abstract model must consist of relatively few canonical concepts. The restriction mechanism depends on the ratio between the number of canonical concepts and the number of elaborations they can have being low. That is, it wants to see few concepts with (possibly) many elaborations. If there are more concepts and fewer elaborations, i.e., if the ratio gets closer to 1, the restriction mechanism becomes less useful. Similarly, the fact that the abstract model consists of a few canonical concepts which can be in different states, mentioned rather lightly in Chapter V, is key here. The reason for both of these dependencies is that restrictivity is defined basically in terms of elaborations and states. If, in a particular domain, the expert model contains many related concepts which are still sufficiently distinct that they are not mostly elaborations of a few concepts; i.e., if the model contains groups of closely related entities that are used in different ways by the deductive processes of the model (all of the elaborations of a concept are used in the same way by the deductive procedures of the model), there is nothing much to restrict anything else. For example, if there were five totally separate kinds of (PEOPLE) in a consultant's model, the fact that there were, say, an elaboration of (PEOPLE-1) in the TOWARD would not restrict the use of (PEOPLE-2)...(PEOPLE-5) in the least. Since these are supposedly closely related concepts, there would be a good chance of their all being used in the same close interaction group. In this case, the restriction mechanism built into the TOWARD would do little to prevent the possibility of exhaustive search through the elaborations of (PEOPLE-2)...(PEOPLE-5) in order to so focussing or reason about the interaction group. Therefore, if there are a great many separate entities in an expert model which are closely related in their deductive uses, but neither elaborations nor states of each other, a focussing control structure based on restriction can become very inefficient. We will see later that this broad type of model structure is in other ways not suitable for focussing.

The analysis of assumption (2) begins very much like that for assumption (1): it would seem that the more input concepts there are, the more advantageous it is to use focussing. Indeed, this is the case so long as the input has certain characteristics. I mentioned earlier that the focussing mechanism is able to get away with relatively few abstracters and structural transformations because the input is canonicalized. We

remember that the job of the canonicalization phase is to substitute standard synonyms for groups of input concepts, and standard constructs for groups of input concept structures. Furthermore, it was stipulated that the canonicalization phase could only make substitutions in cases where no information was lost. That is, it must be possible to define certain variations in wording and sentence structure in the input as not affecting the meaning of the sentence as far as the expert system is concerned. If this is possible, the focussing mechanism can reduce the number of different input objects it has to worry about without decreasing the flexibility of the input it can handle.

Now, even without canonicalization, the restriction mechanism greatly reduces the number of abstracters and transformations that can be applied at any given time during the focussing effort. However, note that the focussing mechanism is rather vulnerable when it comes to sifting through abstracters and (especially) the various possible input constructs. Any increase in the average number of abstracters attached to a concept will be passed directly on to the focussing mechanism, assuming that the average "restriction factor", i.e., the average number of abstracters screened by a single restriction stays constant (which it probably would, because it depends on the elaboration structure of the concepts of the abstract model, not the input, as we saw in the previous paragraph). Still, this is not too bad. If the number of possible input constructs increases, the situation is somewhat worse because the number of structural transformations must be increased to keep pace. That is, if two input constructs, each meaning something slightly different, were introduced where there used to be one, structural transformations would have to be brought in to allow focussing onto the new constructs and perhaps from one construct to the other. The problem with this is that every time a structural transformation is done, it opens up the possibility of gathering new match candidates via the object matching mechanism, which opens the possibility of suggesting new focussing transformations on the basis of those new candidates. The current focussing mechanism relies heavily on the fact that there are only seven structural transformations, only four of which change the object of the construct. If this number were increased, the possibility of inefficiency due to making fruitless structural transformations would increase. Therefore, if canonicalization is not possible in a domain, it can make the focussing methodology much more inefficient--not fatally so, but enough to be uncomfortable if the problem were bad enough.

It seems to me that this problem would arise in domains in which the nuance of each word or sentence structure is important. This is because canonicalization cannot be

used to limit the number of constructs which the system must deal with--there is no way to make substitutions without losing valuable information. I think that there are many domains which, because of standardized terminology and style of presentation are not greatly influenced by the particular nuance of a word or sentence structure. However, in domains in which much is inferred from the *way* something is said, the need for sensitivity to nuance is apparent. I think that psychiatry, politics and diplomacy, and story-understanding are examples of such domains. Of course, these are extreme examples: a medical system that is meant to deal directly with patients (rather than with doctors, as the current ones are) might be on the border line. The system builder can easily judge whether focussing would be inappropriate by keeping tabs on the number of abstracters and structural transformations that are necessary to deal with his input. If they go beyond a given efficiency limit, he must canonicalize. If canonicalization is not feasible, he must choose another method (trading off some other "service", as we will see below).

The third assumption simply says that there is such a thing as the modelled system, and, as the locus of the current state of the mapping under construction, that it is the place to look for feedback information to control the mapping process. Also, it implies that since the modelled system is a reasoning environment, it is liable to be a rather complex data structure (since all of the program's problem-solving goes on it). This means that a separate data structure will be necessary for holding the feedback information so that the mapping procedure doesn't have to sift through the whole map in order to find the information it wants. I don't think that there are any subtle limitations built into this assumption. However, it does make it clear that focussing, as used in Alfred, does depend on there being some well organized cache of the associations successfully completed so far. This raises the question of whether it is worthwhile building such a cache *just* for doing focussing. That is, in reformulation systems, the modelled system must be built in order for the expert deductive procedures to work on the problem (see assumption (4))--no matter what connection mechanism is used. Therefore, the cache is ready-made in reformulation systems. In other systems, say, those in which the whole problem-solving effort consists of finding a path through a deductive network (e.g, [Rieger]), or those in which evidence is gathered in order to select a single self-contained problem-solving unit (e.g., medical diagnosis a la MYCIN [Shortliffe]), there may be no need to retain interim associations. In these cases, the cache would have to be specially built and maintained for focussing to be used. Whether or not this is desirable again depends on the other services needed by the system builder. The necessary trade-offs will be discussed later.

Assumption (4) again raises a possibly non-obvious point. The deductive procedures of the abstract model in a reformulation domain are designed to work on the relatively few canonical concepts of the abstract model (or the input structures that have been associated with them). As such, the deductive procedures are relatively few and general compared with those of problem-solving methods in which many of the concepts may require separate deductive procedures which are local to them and the concepts they directly influence (as in frames, see [Rubin], for example). This relative centralization of deductive knowledge (for example, in Alfred it is confined to eight modelling entities, only one or two of which are used per close interaction group) has the important consequence that the effects of a transformation made on one entity can be easily assessed for other entities of interest. For example, if a transformation is made on one entity in a FLOW, the effects of it will be felt by all of the other entities of the FLOW: if the level of detail is changed, their level of detail must be changed; if a particular elaboration is used, they must be modelled by elaborations whose edges connect up properly; if a particular restriction caused limiting of the options for the first transformation, it must present the same limit for the rest of the transformations of the FLOW. The point is that this is what *defines* the focussing rules for FLOW. The level of detail and restriction rules of Chapter VII were defined on the basis of what changes need to be propagated for the sake of consistency in the deductive procedures attached to each specific modelling entity. This is quite easy given the few general procedures associated with the modelling entities in Alfred. However, contrast this with the distributed control philosophy of frames. Here each frame is responsible for doing whatever transformations are necessary on the particular concepts it deals with, and for calling future frames on the basis of the results of these transformations. Nothing else in the system is *supposed* to worry about what goes on in the individual frames: this is basic to the frame philosophy (proposed frame systems resort to awkward message passing schemes when they need to communicate this kind of information--see [McDermott]). This makes it much harder to figure out and implement the appropriate propagation of focussing transformations; there are no general rules and no small set of deductive procedures which can use them. As I have said previously, I think that this difference is not so much a matter of the characteristics of the domain, but rather of the way in which the problem-solving methodology is conceived (compare some of the vision systems in [Winston] with that of [Marr and Nishihara]). Therefore, it is another question of design trade-off. The system builder may either opt for the central control and piece-wise independence of focussing or for the distributed control of frames with its possibility of separately handling the characteristics of each input and expert concept. Again, more on this kind of trade-off later.

The fifth assumption is of course the basic reason for having the complete mapping phase occur before analysis and debugging begins. That is, it is assumed that the input problem cannot be properly attacked until a sufficient amount of the problem description (where "sufficient" is defined by the theory under consideration) has been modelled. This has the important consequence that the focussing methodology is not designed to accomodate significant remapping efforts (short of trying to select a whole new theory to work on the problem). Alfred does not expect that once the modelled system has been built, i.e., once the ACCOUNT-FOR theme has been completed, the diagnosis and solution-suggesting parts of the program will request that a portion of the problem description be mapped over again in a different way. Either a diagnosis is made and a solution suggested, or the whole effort for that theory has to be scrapped--there are no half-way measures. This is why the backup facilities of the focussing mechanism are so limited. In fact, the only ways that the program can back up and try an alternative are: within a trend, via the PURPOSE mechanism; or between trends, via the trend-changing mechanism.

Actually, the PURPOSE mechanism isn't really a backup facility at all. It is meant to be used for verifying or gathering information for a suggested transformation within a trend. If the verification or information gathering effort turns up something which prohibits the use of the suggested transformation, that transformation can be discarded and another one tried. However, there is no notion of discovering something in a PURPOSE which would cause, say, the last three transformations to have to be changed. The design of the PURPOSE mechanism depends heavily on the fact that it is used within a trend, since this is what ensures that only a few types of PURPOSE subgoals will be necessary, and that the subgoals will be set up to process only specific structures for specific reasons. Therefore, the PURPOSE's could never be expanded into a general backup mechanism.

The only real backup capability, the only way to backup over several transformations and try again, is in the trend change mechanism. This is called in in the true spirit of backup facilities: when the system enters an "error" state in which it cannot continue. For the focussing methodology, this always means one of two things--either there's no piece to set up to continue the map construction process, or there's no way to construct a mapping for a piece that has been set up. The only problem with this mechanism is that it is very hard to utilize the information that has already been modelled in the trend when backup is necessary. This is because, as discussed in Chapter VI, the

reasoning that goes into setting up and focussing a particular piece depends on the dynamic state of the theme, trend, and edge (mostly the trend). By the time it is discovered that the trend is on the wrong track, this state has probably changed several times, *in response to the needs of other set up and focussing problems.* These needs are dependent on both the abstract model and the problem description, in short, on the way the previous parts of the problem description have been modelled. Therefore it is very hard to construct general rules for reasoning about what should be saved if a certain piece cannot be set up or matched--there is no general method for telling which previous association effort led to the particular constraint which is blocking the setup or match, and what the consequences of removing that constraint would be for the rest of the close interaction group. This is the other side of the trade-off of insisting on independence of the knowledge chunks and generality of the focussing procedure: the interaction knowledge needed to assess the consequences of a transformation is built into the focussing procedure rather than being openly represented in each chunk. There are of course exceptions in which it is possible to infer the general consequences of a focussing decision--the "level lapse" concept is one that I thought of which is a general method for any trend, but it is only useful in a very particular set of circumstances.

The control of connection by a feedback process which examines the features of something being built will, I think, always have this difficulty with backup. It is of course possible that someone will think of a good way to reason about procedures which change and interact through a common database, but this isn't to be expected soon. Until then, problem-solving methods which rely on a great deal of hypothesize-and-test strategy will be hard to implement with focussing[1]. For example, I think that medical diagnosis, as discussed in [Miller], would be difficult to implement via focussing for this reason. It is of course possible to imagine schemes which could use focussing before or after some backup-oriented connection mechanism (like the pattern-directed invocation style of PLANNER and Conniver [Sussman and McDermott]), thus getting the best of both worlds, but this issue has not been explored in this thesis at all.

Assumption (6) is based on the fact that in reformulation systems, *structures* of abstract concepts are being matched with *structures* of input concepts. Furthermore, in

[1] I might just say that there is a certain amount of evidence that hypothesize-and-test strategies can be changed into more direct methods which work more efficiently--at least in some domains (see [Waltz], [Sussman and McDermott]).

accordance with assumption (1), the concepts of the expert model are abstract and few. Thus they can, and indeed must, be able to be structured in a variety of ways to appropriately model input concept structures. All of this means that it is unrealistic to expect abstracter-type mappings, i.e., direct entity-to-entity associations, between the constructs of the model and the constructs of the input. Certainly there will be mappings between individual abstract concepts and the (presumably much larger) set of input concepts they can be associated with--otherwise map construction would be impossible. However, the real expert mapping tricks which relate whole model structures to whole input structures (say, at the level of a whole SMOOTH for a particular case, or a small FLOW) will be distributed much more sparsely throughout the model. This is the whole motivation for having a powerful focussing transformation mechanism which uses abstracters only as the terminal processing elements. That is, the mapping engine in Alfred is designed to work abstract concept structures into a form which is in concept-to-concept correspondence with the input structure. The individual concept abstracters can then take over and make the match (or show that no match is possible). When abstracters do exist for the larger structures, they are automatically used as they should be--to increase the efficiency of the mapping process. But, there is no dependence on their being there.

There is also the usual limitation part of this assumption: if the characteristics of the problem-solving method of a proposed expert system are not those described above, the power (and the bother) of an extensive focussing method is not necessary. This is a somewhat tricky issue. On the one hand, there is my earlier premise that most expert systems avoid some of the more interesting aspects of expertise application (like being able to handle more freely expressed input and being able to work at various levels of detail) because their connection mechanisms are not powerful enough (MYCIN is a good example). On the other hand, there is the hypothesis I discussed in Chapter II that some problem-solving techniques really do use a broader, shallower model consisting of a relatively large number of not extremely abstract concepts. With these methods, it is reasonable to expect that there will be fewer possible containing structures for each expert concept, and that the structures that do exist will be closer in form to the input structures. This means that many more of the concept structures in the expert model are liable to have specific interpretations, to mean something special to the expert, because they stand for particular input constructs that are known to be of interest. The effect of this is to make it more likely that the expert model has direct mapping information available for many if not all of the expert concept structures of interest. Another way of

saying this is that there is a strong preconceived notion of which concept structures are likely to be of interest and how they are likely to be mapped into the input.

I am thinking particularly of the frame approach of the recognitionists, in which the concepts of the expert model are formed by generalizing over a class of input objects, and the straight-shooter approach found in, say, MYCIN [Shortliffe] and PROCTOR [Bosyj] in which the domain is carefully digested in order to identify the expert concept structures that are likely to be needed to solve problems in the domain. Recognition systems like those proposed by [Fahlman (Dec., 1973)] and [Kuipers] are composed of frames which have procedures that know what to do with the objects that are mapped into the class of objects defined by the frame prototype. Furthermore, proposals like those of Fahlman and Kuipers (and [Minsky], for that matter) augur relatively broad, shallow structures of frames in which the matching rules for a particular prototype are well known. (The frame prototype is itself just one of the specific structures of expert concepts specially known to the system that I have been talking about.) That is, the possible mappings between the prototype and the class of input objects it can represent are well known. Furthermore, deviations from expectations are handled in a pre-planned manner by other frames. The straight-shooter systems (MYCIN is again an excellent example) are similarly structured, except that the frames are replaced by more passive chunks which are utilized by an interpreter with strong expectations of how the connection effort for a problem should proceed. In both of these methodologies, focussing transformations are not necessary in order to do the mappings, because so much direct mapping information is available. Focussing may be useful in making the system more flexible about the input it can take, and may even be more efficient than the error-directed control scheme proposed for some frame systems, but it is not crucial to the workings of the problem-solver. This puts it into the "service" category discussed below.

There is one final assumption that I must mention, even though it does not come directly from the reformulation model of expertise. The focussing methodology used in Alfred depends very heavily on the fact that there is such a thing as a close interaction group built around a "basic idea". That is, the problem must be segmentable into areas such that there is no significant interaction between the areas except via well defined "ports" (in Alfred's case, the ends of flows). This is the whole basis of the trend or semi-global information structure which carries much of the burden of restricting and controlling the focussing mechanism. Note that I have also slipped in the assumption that any

problem-solving methodology will include the concepts of theme and edge. I think that this is fairly safe. The idea of top level goals and piece-to-piece connectivity information is so universal that it is hard to think of a kind of system in which the appropriate candidates for themes and edges would not be apparent.

The case is somewhat less clear for the trend. Although frame systems like those discussed in [Rubin] and [Kuipers] and straight-shooter systems like MYCIN [Shortliffe] and Brown's system [Brown, A.] have well developed notions of close interaction groups (and basic ideas, or prototypes), the network schemes like those of [Fahlman (May, 1975)] and [Rieger] do not seem to find it necessary. They instead envision much more uniform concept structures whose interactions are determined not by preconceived groupings, but by the needs of a particular goal or query. Also, some problem-solving methods (e.g., see [Hax and Meal]) break problems into hierarchical constraint layers. There may be close interaction within the layers, and even only well defined interaction between them; however, since there are usually only considered to be two or three layers for every problem, the notion of semi-globality is lost. Also, there is no notion of a basic idea around which each layer is built. Therefore, the layers lack the qualities needed for the restriction capabiltiy of the trend.

I find it difficult to imagine a focussing methodology without a trend-like level of control. Given the idea of a close interaction group, it is quite clear where the TOWARD should come from: it is the basic idea of that group, the reason it is being modelled. Furthermore, this immediately defines the notion of PURPOSE's as the subgoals that will be necessary to handle the modelling around this TOWARD (i.e., all of the modelling that must be done within the close interaction group that is not directly concerned with modelling the TOWARD itself). It also defines the initial LEVEL information as consisting of the concepts of the TOWARD, thus enforcing consistency for the level of detail of the close interaction group. Without a close interaction group to provide this structuring and scoping, it is hard to see where these constructs, and more importantly, the kind of control they provide, fit in. This semi-global control certainly doesn't fit into the piece-to-piece scope of the edge information: it relies implicitly on a mechanism which has the scope necessary to propagate it over a whole range of pieces. The theme isn't quite right either, since trend-level information is specifically not meant to be true for the whole model. The top level goals of the theme would have to be artificially scoped to accomodate it. This would make them subgoals rather than top level goals--and in systems without close interaction groups, it is

not clear where these subgoals would come from. Finally, it would be difficult to dynamically configure trend level information (as may be thought of for network schemes) because the whole idea of this information is based on specific restriction rules which are derived from the needs of the deductive procedures of preconceived kinds of close interaction groups (based on modelling entities in Alfred, as I mentioned above).

I would therefore have to say that without a well developed concept of close interaction groups, the focussing methodology, at least as envisioned in this thesis, could not be used.

These are the basic assumptions that were made in designing the focussing approach. They are useful for comparison with the assumptions that may have to be made in order to build future expert systems (based on reformulation or some other problem-solving methodology), and do provide some indication of the strengths and weaknesses of this approach. However, I think that most expert system builders are interested in looking at an implementation methodology in terms of the services it could or could not provide them with, rather than having to figure this out from the degree to which the methodology shares assumptions with their proposed problem-solver. Unfortunately, this kind of analysis is much more speculative than the one above, since it inevitably involves comparison with other approaches. The only intelligent way to view the choice of an implementation methodology is in terms of the design trade-offs it forces as compared to the trade-offs forced by other techniques. There is certainly not likely to be any set of techniques which is best for all or even very many of the possible expertise-application environments that are of interest. However, I think that some are clearly better than others for problem-solving processes with certain characteristics. By choosing a set of design criteria based on these characteristics, it should therefore be possible to select the kind of implementation methodology that is best suited, and then either try to apply it or at least use it for comparative analysis in trying to come up with something better.

With this in mind, I have constructed the table on the following page, which compares a set of services possibly needed by the expert system designer across the sample space of current implementation technology. It must be stressed immediately that, given the lack of experimentation or even objective criteria for judgement, analysis is necessrily speculative. For this reason, I have tried to do nothing more than assign high, low, or medium cost for providing a particular service. These costs are based on the

fundamental biases built into these systems--some services are comfortably within the biases of the system (or are even design goals), in which case they can be provided at low cost; services which go against biases can be provided only at high cost. "Cost" here means that the methodology described would have to be significantly reworked in order to allow it to provide the desired service. The only purpose of the table, then, is to show the areas of strength and weakness in the focussing approach used in Alfred in a wider context than that formed by the assumptions of the method itself.

services \ methods	focussing (ALFRED)	frames (Rubin's system)	production rules (MYCIN)	network (Rieger's system)	OWL (PROCTOR)	pattern-directed (HACKER)
low predictability of input	low cost	high cost	high cost	low cost	medium cost	high cost
high nuance value of individual input concepts	high cost	low cost	high cost	low cost	medium cost	medium cost
many expert concepts	medium cost	high cost	high cost	low cost	medium cost	high cost
high "deductive connectivity"	high cost	low cost	high cost	low cost	medium cost	high cost
high desired independence in putting in parts of the expert model	low cost	high cost	low cost	low cost	high cost	high cost
high desired control over model application	medium cost	low cost	high cost	high cost	low cost	low cost
high need for trial and error processing	high cost	low cost	low cost	?	high cost	low cost
need for modelling at different levels of detail	low cost	medium cost	medium cost	high cost	medium cost	medium cost
highly exportable techniques	medium	low	high	high	?	low
easy to implement with current technology	medium cost	high (?) cost	low cost	very high cost	?	high cost

Table 2. Comparison of Expert System Technologies

A few words of explanation... First of, all "deductive connectivity" refers to the issue discussed in association with assumption (4): some problem-solving methodologies need to take advantage of many concept-specific deduction rules. Therefore, the number of concepts or structures of concepts which mean something special as individual objects to the deductive procedures of the model is liable to increase considerably over that seen in Alfred. Since these deductive procedures are usually based on a relation between several concepts, the "connectivity" or interaction between the concepts in the model increases. Thus, a model which consists of many special deductive procedures and concepts and few general deductive rules has high deductive connectivity. The disease model in [Rubin] has high deductive connectivity.

The cost values for Alfred are pretty predictable given the previous discussion. I have given it "medium"'s for direct extensibility and implementability because it still requires effort to figure out what things should go into the TOWARD, etc. in a given domain, and because an Alfred-type abstract model still requires a considerable amount of thought to build (say, more than a production-rule-based one and less than a pattern-directed-invocation-based one). Frame systems have also been discussed in detail above. Their forte is handling high nuance and high deductive connectivity--at the cost of chunk independence and variability of input. Their extensibility is low because the general rules for building them are much less clear than in other systems. Once these rules have been developed, it is an open question to see how hard they will be to build with current technology. Production systems, on the other hand, are easy to build from the standpoint of existing guidelines, accessible technology, and high independence of the knowledge chunks. However, there has never yet been one that was easily controllable or did not need to take advantage of the fact that it was working in a very limited domain. For this reason, I expect the cost of many desirable large-domain-oriented services to be high. Networks provide the possibility for some services which are very hard to build in terms of other techniques. However, they will be impossible to build efficiently until parallel implementations become feasible. OWL is hard to evaluate because it is incomplete. It stands a good chance of being a highly exportable technique which offers a good cost/capability compromise for some services--if it can ever be thoroughly worked out.

Finally, pattern-directed methods still offer the best possibility for maximum deductive flexibility if the implementer is willing to completely handcraft his system. Few guidelines for implementation have ever been stated, and systems are hard to build. Flexibility with regard to input is also a problem.

The focussing methodology discussed in this thesis therefore represents an attempt at providing a few services which are difficult to get elsewhere, and a package of services which is most appropriate for reformulation-type problem-solving. I believe that, at least in the immediate future, expert system technology will be advanced by methods like this which take advantage of specific desired problem-solver characteristics in order to tailor the group of services they offer to a specialized expert system "market".

Bibliography

Bolinger, Dwight, "The Atomization of Meaning", *Language*, Vol. 41, No.4, Oct.-Dec., 1965.

Bosyj, Michael, *A Program for the Design of Procurement Systems*, MIT Project MAC TR-160, Cambridge, Mass., May, 1976.

Brown, Allen, *Quantitative Knowledge, Causal Reasoning, and the Localization of Failures*, PhD thesis, Electrical Engineering Dept., MIT, Cambridge, Mass., Sept., 1976.

Brown, Gretchen,"Dialogue in OWL", (working paper), MIT, Cambridge, Mass., March, 1974.

Brown, John Seely and Richard Burton, "Multiple Representations of Knowledge for Tutorial Reasoning", in *Representation and Understanding* (Daniel Bobrow and Allen Collins, ed.), Academic Press, New York, 1975.

Carbonell, Jaime and Allan Collins, "Natural Semantics in Artificial Intelligence", *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, Aug., 1973.

Charniak, Eugene, *Toward a Model of Children's Story Comprehension*, MIT AI Lab TR-266, Cambridge, Mass., 1972.

Fahlman, Scott, "A Hypothesis-Frame System for Recognition Problems", MIT AI Lab Working Paper No. 57, Cambridge, Mass., Dec., 1973.

Fahlman, Scott, "A System for Representing and Using Real-World Knowledge", MIT AI Lab Memo No. 331, Cambridge, Mass., May, 1975.

Fahlman, Scott, "The Intersection Problem", MIT AI Lab Working Paper No. 115, Cambridge, Mass., Nov., 1975.

Forrester, Jay, *Industrial Dynamics*, MIT Press, Cambridge, Mass., 1969.

Geertz, Clifford, "On the Nature of Anthropological Understanding", in *Meaning in Anthropology*, (Keith Basso, ed.), New Mexico Press, Albuqurque, 1975.

Goldstein, Ira, *Understanding Simple Picture Programs*, MIT AI Lab TR-294, Sept., 1974.

Gorry, G.A., "The Development of Managerial Models", *Sloan Management Review*, Vol. 12, No. 2, Winter, 1971, pp. 1-16.

Hax, A. and H. Meal, "Hierarchical Integration of Production Planning and Scheduling", in *Studies in Management Sciences*, Vol. I, *Logistics*, (M.A. Geisler, ed.), North Holland-American Elsevier, New York, 1975.

Hewitt, Carl, *Description and Theoretical Analysis (Using Schemata) of PLANNER...*, MIT AI Lab Technical Report No. 259, Cambridge, Mass., April, 1972.

Heidorn, George, "Simulation Programming through Natural Language Dialogue", IBM Research Report RC4535, Yorktown Heights, N.Y., Sept., 1973.

Jarmain, W. Edwin (ed.), *Problems in Industrial Dynamics*, MIT Press, Cambridge, Mass., 1963.

Kuipers, Benjamin, "A Frame for Frames: Representing Knowledge for Recognition", in *Representation and Understanding* (Bobrow and Collins, ed.), Academic Press, New York, 1975.

Krumland, Rand, "Concepts and Structures for a Manager's Modelling System", Dissertation Proposal, Sloan School of Management, MIT, Cambridge, Mass., Oct., 1973.

Kuhn, Thomas, *The Structure of Scientific Revolutions*, University of Chicago Press, Chicago, 1970.

Marr, David and H.K. Nishihara, "Spatial Disposition of Axes in a Generalized Cylinder Representation of Object That Do Not Encompass the Viewer", (whew!), MIT AI Lab Memo No. 341, Cambridge, Mass., Dec., 1975.

Martin, William, "OWL", *Proceedings of the NYU Symposium on Computational Linguistics*, Oct., 1974.

McDermott, Drew, "Very Large PLANNER-Type Data Bases", MIT AI Lab Memo No. 339, Cambridge, Mass., Sept., 1975.

McDermott, Drew and Gerald Sussman, "The Conniver Reference Manual", MIT AI Lab Memo No. 259a, Cambridge, Mass., Jan., 1974.

Miller, Peter, *Strategy Selection in Medical Diagnosis*, MIT Project MAC TR-153, Cambridge, Mass., Sept., 1975.

Minsky, Marvin, "A Framework for Representing Knowledge", MIT AI Lab Memo No. 309, Cambridge, Mass., June, 1974.

Moore, J. and A. Newell, "How Can Merlin Understand?", in *Knowledge and Cognition*, (L. Gregg, ed.), Lawrence Erlbaum Associates, Potomac, Md., 1973.

Moses, Joel, *Symbolic Integration*, MIT Project MAC TR-47, Cambridge, Mass., Dec., 1967.

Naylor, Thomas, *et. al., Computer Simulation Techniques*, John Wiley and Sons, Inc., New York, 1966.

Papert, Seymour, "Teaching Children Thinking", MIT AI Lab Memo No. 247, Cambridge, Mass., October, 1971.

Pennfield, Paul, *MARTHA User's Manual*, MIT Press, Cambridge, Mass., 1971.

Pounds, William, "The Process of Problem Finding", *Industrial Management Review*, Vol. 11, No. 1: Fall, 1969.

Quillian, M. Ross, "The Teachable Language Comprehender: A Simulation Program and Theory of Language", *Communications of the ACM*, Vol. 12, No. 8, Aug., 1969.

Rieger, Chuck, "One System for Two Tasks: A Commonsense Algorithm Memory that Solves Problems and Comprehends Language", MIT AI Lab Working Paper No. 114, Cambridge, Mass., Nov., 1975.

Rubin, Ann D., *Hypothesis Formation and Evaluation in Medical Diagnosis*, MIT AI Lab TR-316, Cambridge, Mass., Jan., 1975.

Schank, Roger, *et. al.*, "MARGIE: Memory Analysis, Response Generation, and Inference on English", *Proceed-ings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, Aug., 1973.

Shortliffe, Edward, *MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Anti-Microbial Therapy Selection*, Stanford University AI Memo No. 251, Oct., 1974.

Sussman, Gerald, *A Computer Model of Skill Acquisition*, American Elsevier, New York, 1975.

Sussman, Gerald and Drew McDermott, "Why Conniving is better than Planning", MIT AI Lab Memo No. 255A, Cambridge, Mass., April, 1972.

Sussman, Gerald and Richard Stallman, "Heuristic Techniques in Computer Aided Circuit Analysis", *IEEE Transactions on Circuits and Systems*, Feb., 1976.

Waltz, David, *Generating Semantic Descriptions from Drawings of Scenes with Shadows*, MIT AI Lab TR-271, Cambridge, Mass., 1972.

Winograd, Terry, *Procedures as a Representation for Data in a Computer Program for Natural Language Understanding*, MIT Project MAC TR-84, Cambridge, Mass., 1971.

Winston, Patrick (ed.), *The Psychology of Computer Vision*, McGraw Hill, New York, 1975.

Woods, William, *Semantics for a Question-Answering System*, PhD Thesis, Harvard University, 1967.

Official Distribution List

Defense Documentation Center
Cameron Station
Alexandria, Va 22314 — 12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, Va 22217 — 2 copies

Office of Naval Research
Code 102IP
Arlington, Va 22217 — 6 copies

Office of Naval Research
Code 200
Arlington, Va 22217 — 1 copy

Office of Naval Research
Code 455
Arlington, Va 22217 — 1 copy

Office of Naval Research
Code 458
Arlington, Va 22217 — 1 copy

Office of Naval Research
Branch Office, Boston
495 Summer Street
Boston, Ma 02210 — 1 copy

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, Il 60605 — 1 copy

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, Ca 91106 — 1 copy

New York Area Office
715 Broadway - 5th floor
New York, N. Y. 10003 — 1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375 — 6 copies

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380 — 1 copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, Ca 92152 — 1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Mathematics Department
Bethesda, Md 20084 — 1 copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350 — 1 copy

Mr. Kin B. Thompson
Technical Director
Information Systems Division (OP-91T)
Office of Chief of Naval Operations
Washington, D. C. 20350 — 1 copy